

This chapter discusses video digitizer components. **Video digitizer components** provide an interface for obtaining digitized video from an analog video source. In QuickTime, the typical client of a video digitizer component is a sequence grabber component (sequence grabber components are described in the chapter “Sequence Grabber Components” in this book). Sequence grabber components use the services of video digitizer components and image compressor components to create a simple interface for making and previewing movies. However, video digitizer components can also operate independently, placing video into a window.

IMPORTANT

Most applications never need to communicate directly with a video digitizer component. It is strongly advised that your application use the sequence grabber component instead; it isolates you from the myriad of details associated with video digitization. ▲

This chapter has been divided into the following major sections:

- “About Video Digitizer Components” presents some general information about video digitizer components.
- “Using Video Digitizer Components” gives details on how you tell the digitizer where to put the data and how to control digitization. It describes a technique for improving performance.
- “Creating Video Digitizer Components” discusses how to create a video digitizer component.
- “Video Digitizer Components Reference” describes the constants, data structures, and functions associated with video digitizer components.
- “Summary of Video Digitizer Components” supplies a summary of the constants, data types, and functions associated with video digitizer components in C and in Pascal.

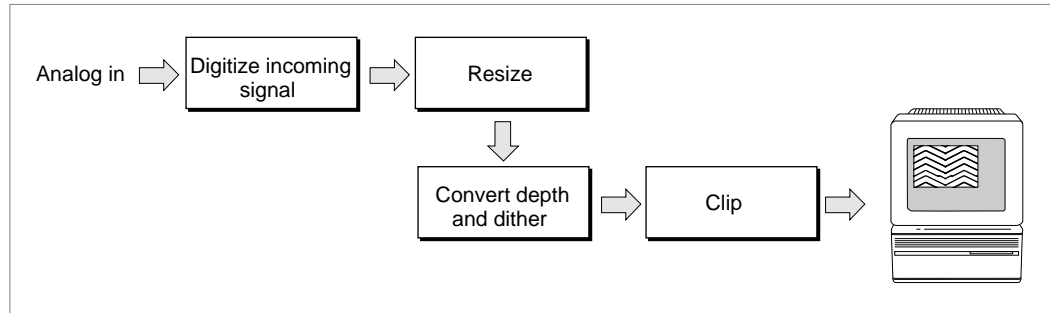
About Video Digitizer Components

Video digitizer components convert video input into a digitized color image that is compatible with the graphics system of a computer. For example, a video digitizer may convert input analog video into a specified digital format. The input may be any video format and type, whereas the output must be intelligible to the Macintosh computer’s display system. Once the digitizer has converted the input signal to an appropriate digital format, it then prepares the image for display by resizing the image, performing necessary color conversions, and clipping to the output window. At the end of this process, the digitizer component places the converted image into a buffer you specify—if that buffer is the current frame buffer, the image appears on the user’s computer screen.

Video Digitizer Components

Figure 8-1 shows the steps involved in converting the analog video signal to digital format and preparing the digital data for display. Some video digitizer components perform all these steps in hardware. Others perform some or all of these steps in software. Others may perform only a few of these steps—in which case, it is up to the program that is using the video digitizer to perform these tasks.

Figure 8-1 Basic tasks of a video digitizer



Video digitizer components resize the image by applying a transformation matrix to the digitized image. Your application specifies the matrix that is applied to the image. Matrix operations can enlarge or shrink an image, distort the image, or move the location of an image. The Movie Toolbox provides a set of functions that make it easy for you to work with transformation matrices. See the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for more information about matrix operations.

Before the digitized image can be displayed on your computer, the video digitizer component must convert the image into an appropriate color representation. This conversion may involve dithering or pixel depth conversion. The digitizer component handles this conversion based on the destination characteristics you specify.

Video digitizer components may support clipping. Digitizers that do support clipping can display the resulting image in regions of arbitrary shapes. See the next section for a complete discussion of the techniques that digitizer components can use to perform clipping.

Types of Video Digitizer Components

Video digitizer components fall into four categories, distinguished by their support for clipping a digitized video image:

- basic digitizers, which do not support clipping
- alpha channel digitizers, which clip by means of an alpha channel
- mask plane digitizers, which clip by means of a mask plane
- key color digitizers, which clip by means of key colors

Basic video digitizer components are capable of placing the digitized video into memory, but they do not support any graphics overlay or video blending. If you want to perform these operations, you must do so in your application. For example, you can stop the digitizer after each frame and do the work necessary to blend the digitized video with a graphics image that is already being displayed. Unfortunately, this may cause jerkiness or discontinuity in the video stream. Other types of digitizers that support clipping make this operation much easier for your application.

Alpha channel digitizer components use a portion of each display pixel to represent the blending of video and graphical image data. This part of each pixel is referred to as an **alpha channel**. The size of the alpha channel differs depending upon the number of bits used to represent each pixel. For 32 bits per pixel modes, the alpha channel is represented in the 8 high-order bits of each 32-bit pixel. These 8 bits can define up to 256 levels of blend. For 16 bits per pixel modes, the alpha channel is represented in the high-order bit of the pixel and defines one level of blend (on or off).

Mask plane digitizer components use a pixel map to define blending. Values in this mask correspond to pixels on the screen, and they define the level of blend between video and graphical image data.

Key color digitizer components determine where to display video data based upon the color currently being displayed on the output device. These digitizers reserve one or more colors in the color table; these colors define where to display video. For example, if blue is reserved as the key color, the digitizer replaces all blue pixels in the display rectangle with the corresponding pixels of video from the input video source.

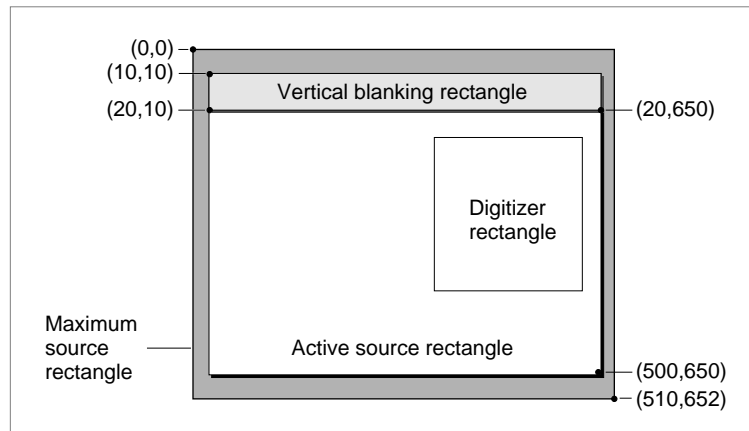
Source Coordinate Systems

Your application can control what part of the source video image is extracted. The digitizer then converts the specified portion of the source video signal into a digital format for your use. Video digitizer components define four areas you may need to manipulate when you define the source image for a given operation. These areas are

- the maximum source rectangle
- the active source rectangle
- the vertical blanking rectangle
- the digitizer rectangle

Figure 8-2 shows the relationships between these rectangles.

Figure 8-2 Video digitizer rectangles



The **maximum source rectangle** defines the maximum source area that the digitizer component can grab. This rectangle usually encompasses both the vertical and horizontal blanking areas. The **active source rectangle** defines that portion of the maximum source rectangle that contains active video. The **vertical blanking rectangle** defines that portion of the input video signal that is devoted to vertical blanking. This rectangle occupies lines 10 through 19 of the input signal. Broadcast video sources may use this portion of the input signal for closed captioning, teletext, and other nonvideo information. Note that the blanking rectangle might not be contained in the maximum source rectangle.

You specify the **digitizer rectangle**, which defines that portion of the active source rectangle that you want to capture and convert.

Using Video Digitizer Components

This section describes how you can control a video digitizer component. It has been divided into the following topics:

- “Specifying Destinations” discusses how you tell the digitizer where to put the converted video data.
- “Starting and Stopping the Digitizer” discusses how you control digitization.
- “Multiple Buffering” describes a technique for improving performance.
- “Obtaining an Accurate Time of Frame Capture” tells how the sequence grabber usually supplies video digitizers with a time base. This time base lets your application get an accurate time for the capture of any specified frame.

Specifying Destinations

Video digitizer components provide several functions that allow applications to specify the destination for the digitized video stream produced by the digitizer component. You have two options for specifying the destination for the video data stream in your application.

The first option requires that the video be digitized as RGB pixels and placed into a destination pixel map. This option allows the video to be placed either onscreen or offscreen, depending upon the placement of the pixel map. Your application can use the `VDSetPlayThruDestination` function (described on page 8-35) to set the characteristics for this option. Your application can use the `VDPreflightDestination` function (described on page 8-36) to determine the capabilities of the digitizer. All video digitizer components must support this option.

The second option uses a global boundary rectangle to define the destination for the video. This option always results in onscreen images and is useful with digitizers that support hardware direct memory access (DMA) across multiple screens. The digitizer component is responsible for any required color depth conversions, image clipping and resizing, and so on. Your application can use the `VDSetPlayThruGlobalRect` function (described on page 8-39) to set the characteristics for this option. Your application can use the `VDPreflightGlobalRect` function (described on page 8-40) to determine the capabilities of the digitizer. Not all video digitizer components support this option.

Starting and Stopping the Digitizer

You can control digitization on a frame-by-frame basis in your application. The `VDGrabOneFrame` function (described on page 8-54) digitizes a single video frame. All video digitizer components support this function.

Alternatively, you can use the `VDSetPlayThruOnOff` function (described on page 8-53) to enable or disable digitization. When digitization is enabled, the video digitizer component places video into the specified destination continuously. The application stops the digitizer by disabling digitization. This function can be used with both destination options. However, not all video digitizer components support this function.

Multiple Buffering

You can improve the performance of frame-by-frame digitization by using multiple destination buffers for the digitized video. Your application defines a number of destination buffers to the video digitizer component and specifies the order in which those buffers are to be used. The digitizer component then fills the buffers, allowing you to switch between the buffers more quickly than your application otherwise could. In this manner, you can grab a video sequence at a higher rate with less chance of data loss. This technique can be used with both destination options.

You define the buffers to the digitizer by calling the `VDSetupBuffers` function (described on page 8-54). The `VDGrabOneFrameAsync` function (described on page 8-56) starts the process of grabbing a single video frame. The `VDDone` function (described on page 8-58) allows you to determine when the digitizer component has finished a given frame.

Obtaining an Accurate Time of Frame Capture

The sequence grabber typically gives video digitizers a time base so your application can obtain an accurate time for the capture of any given frame. Applications can set the digitizer's time base by calling the `VDSetTimeBase` function, which is described on page 8-51.

Creating Video Digitizer Components

Video digitizer components are the most convenient mechanism for presenting new sources of video data to QuickTime. For example, if you are developing special-purpose video hardware that digitizes video images from a previously unsupported source device, you should create a video digitizer component so that applications or sequence grabber components can obtain data from your device.

Refer to the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox* for a general discussion of how to create a component.

The remaining topics in this section discuss issues you should consider when creating a video digitizer component.

Apple has defined a functional interface for video digitizer components. For information about the functions your digitizer component must support, see “Video Digitizer Component Functions” beginning on page 8-23.

You can use the following enumerators to refer to the request codes for each of the functions that your component must support.

```
enum {
    /* video digitizer interface */
    kSelectVDGetMaxSrcRect          = 0x1, /* VDGetMaxSrcRect (required) */
    kSelectVDGetActiveSrcRect      = 0x2, /* VDGetActiveSrcRect
                                           (required) */
    kSelectVDSetDigitizerRect      = 0x3, /* VDSetDigitizerRect
                                           (required) */
    kSelectVDGetDigitizerRect      = 0x4, /* VDGetDigitizerRect
                                           (required) */
    kSelectVDGetVBlankRect         = 0x5, /* VDGetVBlankRect (required) */
    kSelectVDGetMaskPixMap         = 0x6, /* VDGetMaskPixMap */
    kSelectVDGetPlayThruDestination = 0x8, /* VDGetPlayThruDestination
                                           (required) */

    kSelectVDUseThisCLUT           = 0x9, /* VDUseThisCLUT */
    kSelectVDSetInputGammaValue    = 0xA, /* VDSetInputGammaValue */
    kSelectVDGetInputGammaValue    = 0xB, /* VDGetInputGammaValue */
    kSelectVDSetBrightness         = 0xC, /* VDSetBrightness */
    kSelectVDGetBrightness         = 0xD, /* VDGetBrightness */
    kSelectVDSetContrast           = 0xE, /* VDSetContrast */
    kSelectVDSetHue                = 0xF, /* VDSetHue */
    kSelectVDSetSharpness          = 0x10, /* VDSetSharpness */
    kSelectVDSetSaturation          = 0x11, /* VDSetSaturation */
    kSelectVDGetContrast           = 0x12, /* VDGetContrast */
    kSelectVDGetHue                = 0x13, /* VDGetHue */
    kSelectVDGetSharpness          = 0x14, /* VDGetSharpness */
    kSelectVDGetSaturation          = 0x15, /* VDGetSaturation */
    kSelectVDGrabOneFrame          = 0x16, /* VDGrabOneFrame
                                           (required) */

    kSelectVDGetMaxAuxBuffer       = 0x17, /* VDGetMaxAuxBuffer */
    kSelectVDGetDigitizerInfo      = 0x19, /* VDGetDigitizerInfo
                                           (required) */

    kSelectVDGetCurrentFlags       = 0x1A, /* VDGetCurrentFlags
                                           (required) */

    kSelectVDSetKeyColor           = 0x1B, /* VDSetKeyColor */
    kSelectVDGetKeyColor           = 0x1C, /* VDGetKeyColor */
    kSelectVDAAddKeyColor          = 0x1D, /* VDAAddKeyColor */
    kSelectVDGetNextKeyColor       = 0x1E, /* VDGetNextKeyColor */
    kSelectVDSetKeyColorRange      = 0x1F, /* VDSetKeyColorRange */
    kSelectVDGetKeyColorRange      = 0x20, /* VDGetKeyColorRange */
    kSelectVDSetDigitizerUserInterrupt = 0x21,
                                           /* VDSetDigitizerUserInterrupt */
}
```

Video Digitizer Components

```

kSelectVDSetInputColorSpaceMode      = 0x22, /* VDSetInputColorSpaceMode */
kSelectVDGetInputColorSpaceMode      = 0x23, /* VDGetInputColorSpaceMode */
kSelectVDSetClipState                = 0x24, /* VDSetClipState */
kSelectVDGetClipState                = 0x25, /* VDGetClipState */
kSelectVDSetClipRgn                  = 0x26, /* VDSetClipRgn */
kSelectVDClearClipRgn                = 0x27, /* VDClearClipRgn */
kSelectVDGetCLUTInUse                = 0x28, /* VDGetCLUTInUse */
kSelectVDSetPLLFilterType            = 0x29, /* VDSetPLLFilterType */
kSelectVDGetPLLFilterType            = 0x2A, /* VDGetPLLFilterType */
kSelectVDGetMaskandValue             = 0x2B, /* VDGetMaskandValue */
kSelectVDSetMasterBlendLevel         = 0x2C, /* VDSetMasterBlendLevel */
kSelectVDSetPlayThruDestination      = 0x2D, /* VDSetPlayThruDestination */
kSelectVDSetPlayThruOnOff            = 0x2E, /* VDSetPlayThruOnOff */
kSelectVDSetFieldPreference          = 0x2F, /* VDSetFieldPreference
                                           (required) */
kSelectVDGetFieldPreference          = 0x30, /* VDGetFieldPreference
                                           (required) */
kSelectVDPreflightDestination        = 0x32, /* VDPreflightDestination
                                           (required) */
kSelectVDPreflightGlobalRect         = 0x33, /* VDPreflightGlobalRect */
kSelectVDSetPlayThruGlobalRect       = 0x34, /* VDSetPlayThruGlobalRect */
kSelectVDSetInputGammaRecord         = 0x35, /* VDSetInputGammaRecord */
kSelectVDGetInputGammaRecord         = 0x36, /* VDGetInputGammaRecord */
kSelectVDSetBlackLevelValue          = 0x37, /* VDSetBlackLevelValue */
kSelectVDGetBlackLevelValue          = 0x38, /* VDGetBlackLevelValue */
kSelectVDSetWhiteLevelValue          = 0x39, /* VDSetWhiteLevelValue */
kSelectVDGetWhiteLevelValue          = 0x3A, /* VDGetWhiteLevelValue */
kSelectVDGetVideoDefaults            = 0x3B, /* VDGetVideoDefaults */
kSelectVDGetNumberOfInputs           = 0x3C, /* VDGetNumberOfInputs */
kSelectVDGetInputFormat              = 0x3D, /* VDGetInputFormat */
kSelectVDSetInput                    = 0x3E, /* VDSetInput */
kSelectVDGetInput                    = 0x3F, /* VDGetInput */
kSelectVDSetInputStandard            = 0x40, /* VDSetInputStandard */
kSelectVDSetupBuffers                = 0x41, /* VDSetupBuffers */
kSelectVDGrabOneFrameAsync           = 0x42, /* VDGrabOneFrameAsync */
kSelectVDDone                        = 0x43, /* VDDone */
kSelectVDSetCompression              = 0x44, /* VDSetCompression */
kSelectVDCompressOneFrameAsync       = 0x45, /* VDCompressOneFrameAsync */
kSelectVDCompressDone                = 0x46, /* VDCompressDone */
kSelectVDReleaseCompressBuffer       = 0x47, /* VDReleaseCompressBuffer */
kSelectVDGetImageDescription         = 0x48, /* VDGetImageDescription */
kSelectVDResetCompressSequence       = 0x49, /* VDResetCompressSequence */
kSelectVDSetCompressionOnOff         = 0x4A, /* VDSetCompressionOnOff */

```


Video Digitizer Components

```

kSelectVDGetCompressionTypes      = 0x4B, /* VDGetCompressionTypes */
kSelectVDSetTimeBase              = 0x4C, /* VDSetTimeBase */
kSelectVDSetFrameRate             = 0x4D, /* VDSetFrameRate */
kSelectVDGetDataRate              = 0x4E, /* VDGetDataRate */
kSelectVDGetSoundInputDriver      = 0x4F, /* VDGetSoundInputDriver */
kSelectVDGetDMADepths             = 0x50, /* VDGetDMADepths */
kSelectVDGetPreferredTimeScale    = 0x51, /* VDGetPreferredTimeScale */
kSelectVDReleaseAsyncBuffers      = 0x52, /* VDReleaseAsyncBuffers */
};

```

Component Type and Subtype Values

Apple has defined a type value for video digitizer components. All video digitizer components have a component type value of 'vdig'. You can use the following constant to specify the component type value.

```
#define videoDigitizerComponentType = 'vdig'
```

There are no special conventions applied to the subtype value of video digitizer components.

Required Functions

Video digitizer components support a rich functional interface that can accommodate devices with quite varied capabilities. To relieve you from having to support irrelevant functions, Apple has made several video digitizer functions optional.

At a minimum, your video digitizer component must support the following functions:

VDGetActiveSrcRect	VDGetCurrentFlags
VDGetDigitizerInfo	VDGetDigitizerRect
VDGetFieldPreference	VDGetInput
VDGetInputFormat	VDGetMaxSrcRect
VDGetNumberOfInputs	VDGetPlayThruDestination
VDGetVBlankRect	VDGetVideoDefaults
VDGrabOneFrame	VDPreflightDestination
VDSetDigitizerRect	VDSetFieldPreference
VDSetInput	VDSetInputStandard
VDSetPlayThruDestination	

All of these functions are required for all video digitizer components.

Optional Functions

Based on the type of device your component supports, you may have to implement functions other than those listed in “Required Functions,” and you may have to set some of your component’s capability flags. Read this section to learn which additional functions your component needs to support and how to set your capability flags properly.

If your component does not support a particular function, be sure to return a result code value of `digiUnimpErr`.

Note

Hardware support for the simultaneous capture and display of frames on the screen is called *playthrough* in these sections. ♦

Frame Grabbers Without Playthrough

Suppose your video digitization hardware grabs frames but cannot simultaneously display the frames on the screen. Suppose also that your hardware supplies the grabbed frames in QuickDraw pixel maps at specific pixel depths (say, 16 and 32 bits per pixel). For details on QuickDraw pixel maps, see the chapter “Basic QuickDraw” in *Inside Macintosh: Imaging*.

In this case, you should set the following component capability flags:

```
digiOutDoes16    Set this flag to 1.
digiOutDoes32    Set this flag to 1.
                  Set other depth flags to 0.
digiOutDoesHWPlayThru
                  Set this flag to 0.
digiOutDoesDMA    Set this flag to 0.
```

If your component can operate asynchronously, you should also set the following flag:

```
digiOutDoesAsyncGrabs
                  Set this flag to 1 if your component can operate asynchronously.
```

Frame grabbers that support asynchronous operation must support the following optional functions:

```
VDDone                VDGrabOneFrameAsync
VDReleaseAsyncBuffers  VDSetupBuffers
```

Frame Grabbers With Hardware Playthrough

If your frame grabber hardware provides support for playing the captured images directly, you need to support one additional function beyond those discussed in “Frame Grabbers Without Playthrough.” The `VDSetPlayThruOnOff` function (described on page 8-53) allows the application to turn playthrough on and off.

You should also set the `digiOutDoesHWPlayThru` capability flag (described on page 8-18) to 1. In addition, be sure to use the `gdh` field in the digitizer information structure to identify your component's display device. For details on the video digitizer information structure, see page 8-20.

Key Color and Alpha Channel Devices

As a further elaboration on a basic frame grabber, your device could support the display or mixing of output data via an alpha channel or through the use of key colors (see “Types of Video Digitizer Components” on page 8-5 for more information about alpha channels and key colors). In either case, image data cannot be read directly from the screen. Therefore, you must set the `digiOutDoesUnreadableScreenBits` capability flag to 1. For more on the video digitizer capability flags, see “Capability Flags” beginning on page 8-14.

Your component must load its alpha channel or fill in the key color whenever playthrough is enabled or when the destination changes.

Compressed Source Devices

You may create a video digitizer component that supports a device that delivers compressed image data. In this case, your component is not capable of displaying the data directly.

Your component should set the following capability flags:

`digiOutDoesCompress`

Set this flag to 1.

`digiOutDoesCompressOnly`

Set this flag to 1 if your component cannot display the images directly.

`digiOutDoesPlayThruDuringCompress`

Set this flag to 1 if your component cannot display the images directly.

In addition, frame grabbers that support compressed source devices must support the following optional functions:

<code>VDCompressDone</code>	<code>VDCompressOneFrameAsync</code>
<code>VDGetCompressionTypes</code>	<code>VDGetDataRate</code>
<code>VDGetImageDescription</code>	<code>VDResetCompressSequence</code>
<code>VDSetCompression</code>	<code>VDSetCompressionOnOff</code>
<code>VDSetFrameRate</code>	<code>VDSetTimeBase</code>

If your hardware generates compressed data that cannot be decompressed by any standard QuickTime image decompressor components, be sure to provide an appropriate decompressor component so that the data you provide can be displayed.

Video Digitizer Components Reference

The following sections describe the constants, data structures, and functions that are specific to video digitizer components.

Constants

This section provides details on the video digitizer component's capability and current flags.

Capability Flags

Video digitizer components report their capabilities to your application by means of capability flags. These flags are formatted as part of the digitizer information structure you obtain by calling the `VDGetDigitizerInfo` function, which is described on page 8-24. There are two sets of flags: one set describes the input capabilities of the video digitizer component; the other describes its output capabilities.

Video digitizer components support the following input capability flags:

`digiInDoesNTSC`

Indicates that the video digitizer supports **National Television System Committee (NTSC)** format input video signals. This flag is set to 1 if the digitizer component supports NTSC video.

`digiInDoesPAL`

Indicates that the video digitizer component supports **Phase Alternation Line (PAL)** format input video signals. This flag is set to 1 if the digitizer component supports PAL video.

`digiInDoesSECAM`

Indicates that the video digitizer component supports **Système Electronique Couleur avec Memoire (SECAM)** format input video signals. This flag is set to 1 if the digitizer component supports SECAM video.

`digiInDoesGenLock`

Indicates that the video digitizer component supports **genlock**; that is, the digitizer can derive its timing from an external time base. This flag is set to 1 if the digitizer component supports genlock.

`digiInDoesComposite`

Indicates that the video digitizer component supports composite input video. This flag is set to 1 if the digitizer component supports composite input.

Video Digitizer Components

`digitInDoesSVideo`

Indicates that the video digitizer component supports **s-video** input video. This flag is set to 1 if the digitizer component supports s-video input.

`digiInDoesComponent`

Indicates that the video digitizer component supports RGB input video. This flag is set to 1 if the digitizer component supports RGB input.

`digitInVTR_Broadcast`

Indicates that the video digitizer component can distinguish between an input signal that emanates from a videotape player and a broadcast signal. This flag is set to 1 if the digitizer component can differentiate between the two different signal types.

`digiInDoesColor`

Indicates that the video digitizer component supports color input. This flag is set to 1 if the digitizer component can accept color input.

`digiInDoesBW`

Indicates that the video digitizer component supports grayscale input. This flag is set to 1 if the digitizer component can accept grayscale input.

Video digitizer components support the following output capability flags:

`digitOutDoes1`

Indicates that the video digitizer component can work with pixel maps that contain 1-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 1-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digitOutDoes2`

Indicates that the video digitizer component can work with pixel maps that contain 2-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 2-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digitOutDoes4`

Indicates that the video digitizer component can work with pixel maps that contain 4-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 4-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digitOutDoes8`

Indicates that the video digitizer component can work with pixel maps that contain 8-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 8-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digitOutDoes16`

Indicates that the video digitizer component can work with pixel maps that contain 16-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 16-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

Video Digitizer Components

`digiOutDoes32`

Indicates that the video digitizer component can work with pixel maps that contain 32-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 32-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoesDither`

Indicates that the video digitizer component supports dithering. If this flag is set to 1, the component supports dithering of colors. If this flag is set to 0, the digitizer component does not support dithering.

`digiOutDoesStretch`

Indicates that the video digitizer component can stretch images to arbitrary sizes. If this flag is set to 1, the digitizer component can stretch images. If this flag is set to 0, the digitizer component does not support stretching.

`digiOutDoesShrink`

Indicates that the video digitizer component can shrink images to arbitrary sizes. If this flag is set to 1, the digitizer component can shrink images. If this flag is set to 0, the digitizer component does not support shrinking.

`digiOutDoesMask`

Indicates that the video digitizer component can handle clipping regions. If this flag is set to 1, the digitizer component can mask to an arbitrary clipping region. If this flag is set to 0, the digitizer component does not support clipping regions.

`digiOutDoesDouble`

Indicates that the video digitizer component supports stretching to quadruple size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request—the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly four times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support stretching to quadruple size.

`digiOutDoesQuad`

Indicates that the video digitizer component supports stretching an image to 16 times its original size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request—the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly 16 times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

Video Digitizer Components

`digiOutDoesQuarter`

Indicates that the video digitizer component can shrink an image to one-quarter of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request—the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly one-quarter of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesSixteenth`

Indicates that the video digitizer component can shrink an image to 1/16 of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request—the digitizer component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly 1/16 of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesRotate`

Indicates that the video digitizer component can rotate an image when displaying the output video. The parameters for the rotation are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can rotate the image. If this flag is set to 0, the digitizer component cannot rotate the resulting image.

`digiOutDoesHorizFlip`

Indicates that the video digitizer component can flip an image horizontally when displaying the output video. The parameters for the horizontal flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

`digiOutDoesVertFlip`

Indicates that the video digitizer component can flip an image vertically when displaying the output video. The parameters for the vertical flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

`digiOutDoesSkew`

Indicates that the video digitizer component can skew an image when displaying the output video. Skewing an image distorts it linearly along only a single axis—for example, drawing a rectangular image into a parallelogram-shaped region. The parameters for the skew operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can skew an image. If this flag is set to 0, the digitizer component does not support this capability.

Video Digitizer Components

`digiOutDoesBlend`

Indicates that the video digitizer component can blend the resulting image with a matte when displaying the output video. The matte is provided by the application by defining either an alpha channel or a mask plane. If this flag is set to 1, the digitizer component can blend. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesWarp`

Indicates that the video digitizer component can warp an image when displaying the output video. Warping an image distorts it along one or more axes, perhaps nonlinearly, in effect “bending” the result region. The parameters for the warp operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can warp an image. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesDMA`

Indicates that the video digitizer component can write to any screen or to offscreen memory. If this flag is set to 1, the digitizer component can use DMA to write to any screen or memory location.

`digiOutDoesHWPlayThru`

Indicates that the video digitizer component does not need idle time in order to display its video. If this flag is set to 1, your application does not need to grant processor time to the digitizer component at normal display speeds.

`digiOutDoesILUT`

Indicates that the video digitizer component supports inverse lookup tables for indexed color modes. If this flag is set to 1, the digitizer component uses inverse lookup tables when appropriate.

`digiOutDoesKeyColor`

Indicates that the video digitizer component supports clipping by means of key colors. If this flag is set to 1, the digitizer component can clip to a region defined by a key color.

`digiOutDoesAsyncGrabs`

Indicates that the video digitizer component can operate asynchronously. If this flag is set to 1, your application can use the `VDSetupBuffers` and `VDGrabOneFrameAsync` functions (described on page 8-54 and page 8-56, respectively).

`digiOutDoesUnreadableScreenBits`

Indicates that the video digitizer may place pixels on the screen that cannot be used when compressing images.

`digiOutDoesCompress`

Indicates that the video digitizer component supports compressed source devices. These devices provide compressed data directly, without having to use the Image Compression Manager. See “Controlling Compressed Source Devices” beginning on page 8-42 for more information about the functions that applications can use to work with compressed source devices.

Video Digitizer Components

`digiOutDoesCompressOnly`

Indicates that the video digitizer component only provides compressed image data; the component cannot provide displayable data. This flag only applies to digitizers that support compressed source devices.

`digiOutDoesPlayThruDuringCompress`

Indicates that the video digitizer component can draw images on the screen at the same time that it is delivering compressed image data. This flag only applies to digitizers that support compressed source devices.

Current Flags

Video digitizer components report their current status to your application by means of flags. These flags are formatted as part of the digitizer information structure that you obtain by calling the `VDGetDigitizerInfo` function (described on page 8-24).

Alternatively, you can obtain these flags by calling the `VDGetCurrentFlags` function (described on page 8-25). There are two sets of flags: one set describes the status of the digitizer with respect to its input signal; the other describes its status with respect to its output.

Video digitizer components report their current status by returning a flags field that contains 1 bit for each of the capability flags (discussed in “Capability Flags” beginning on page 8-14) plus additional flags as appropriate. The digitizer component sets these flags to reflect its current status. When reporting input status, for example, a video digitizer component sets the `digiInDoesGenLock` flag to 1 whenever the digitizer component is deriving its time signal from the input video. When reporting its input capabilities, the digitizer component sets this flag to 1 to indicate that it can derive its timing from the input video.

Video digitizer components report their current input status by returning a flags field that contains a bit for each of the input capability flags (discussed in “Capability Flags” beginning on page 8-14) plus one additional flag.

The additional flag is as follows:

`digiInSignalLock`

Indicates that the video digitizer component is locked onto the input signal. If this flag is set to 1, the digitizer component detects either vertical or horizontal signal lock.

Video digitizer components report their current output status by returning a flags field that contains a bit for each of the output capability flags discussed in “Capability Flags” beginning on page 8-14. The digitizer component sets these flags to reflect its current output status.

Data Types

This section discusses the data structures that are used by video digitizer components and by applications that use video digitizer components.

The Digitizer Information Structure

Your application can retrieve information about the capabilities and current status of a video digitizer component. You call the `VDGetDigitizerInfo` function, described on page 8-24, to retrieve all this information from a video digitizer component. In response, the component formats a digitizer information structure. The contents of this structure fully define the capabilities and current status of the video digitizer component.

Note

If you are interested only in the current status information, you can call the `VDGetCurrentFlags` function, which is described on page 8-25. This function returns the input and output current flags of the video digitizer component. ♦

The `DigitizerInfo` data type defines the layout of the digitizer information structure.

```
struct DigitizerInfo {
    short    vdigType;           /* type of digitizer component */
    long     inputCapabilityFlags; /* input video signal features */
    long     outputCapabilityFlags; /* output digitized video data
                                   features of digitizer component */
    long     inputCurrentFlags;   /* status of input video signal */
    long     outputCurrentFlags;  /* status of output digitized
                                   video information */
    short    slot;               /* for connection purposes */
    GDHandle gdh;                /* for digitizers with preferred
                                   screen */
    GDHandle maskgdh;            /* for digitizers with mask planes */
    short    minDestHeight;      /* smallest resizable height */
    short    minDestWidth;       /* smallest resizable width */
    short    maxDestHeight;      /* largest resizable height */
    short    maxDestWidth;       /* largest resizable width */
    short    blendLevels;        /* number of blend levels supported
                                   (2 if 1-bit mask) */
    long     private;            /* reserved--set to 0 */
};
typedef struct DigitizerInfo DigitizerInfo;
```

Video Digitizer Components

Field descriptions

<code>vdigType</code>	Specifies the type of video digitizer component. Valid values are
<code>vdTypeBasic</code>	Basic video digitizer—does not support any clipping
<code>vdTypeAlpha</code>	Supports clipping by means of an alpha channel
<code>vdTypeMask</code>	Supports clipping by means of a mask plane
<code>vdTypeKey</code>	Supports clipping by means of key colors
<code>inputCapabilityFlags</code>	Specifies the capabilities of the video digitizer component with respect to the input video signal. These flags are discussed in “Capability Flags” beginning on page 8-14.
<code>outputCapabilityFlags</code>	Specifies the capabilities of the video digitizer component with respect to the output digitized video information. These flags are discussed in “Capability Flags” beginning on page 8-14.
<code>inputCurrentFlags</code>	Specifies the current status of the video digitizer with respect to the input video signal. These flags are discussed in “Current Flags” on page 8-19.
<code>outputCurrentFlags</code>	Specifies the current status of the video digitizer with respect to the output digitized video information. These flags are discussed in “Current Flags” on page 8-19.
<code>slot</code>	Identifies the slot that contains the video digitizer interface card.
<code>gdh</code>	Contains a handle to the graphics device that defines the screen to which the digitized data is to be written. Set this field to <code>nil</code> if your application is not constrained to a particular graphics device.
<code>maskgdh</code>	Contains a handle to the graphics device that contains the mask plane. This field is used only by digitizers that clip by means of mask planes.
<code>minDestHeight</code>	Indicates the smallest height value the digitizer component can accommodate in its destination.
<code>minDestWidth</code>	Indicates the smallest width value the digitizer component can accommodate in its destination.
<code>maxDestHeight</code>	Indicates the largest height value the digitizer component can accommodate in its destination.
<code>maxDestWidth</code>	Indicates the largest width value the digitizer component can accommodate in its destination.

Video Digitizer Components

blendLevels

Specifies the number of blend levels the video digitizer component supports.

private

Reserved. Set this field to 0.

The Buffer List Structure

If you are using more than one asynchronous output buffer, you must define the output buffers to the video digitizer component. You define these output buffers by calling the `VDSetupBuffers` function (described on page 8-54). You specify the buffers to that function in a buffer list structure. Note that all the output buffers must be the same size and must accommodate output rectangles of the same dimensions.

The `VdigBufferRecList` data type defines a buffer list structure.

```
struct VdigBufferRecList {
    short          count;    /* number of buffers defined by
                             this structure */
    MatrixRecordPtr matrix; /* tranformation matrix applied to
                             destination rectangles before
                             video image is displayed */
    RgnHandle      mask;     /* clipping region applied to
                             destination rectangle before
                             video image is displayed */
    VdigBufferRec  list[1]; /* array of output buffer
                             specifications */
};
```

Field descriptions

count	Specifies the number of buffers defined by this structure. The value of this field must correspond to the number of entries in the <code>list</code> array.
matrix	Specifies the transformation matrix that is applied to all of the destination rectangles before the video image is displayed. You must specify a matrix. If you do not want to perform any transformations, use the identity matrix.
mask	Specifies a clipping region that is applied to the destination rectangle before the video image is displayed. Note that this region applies to only the first destination buffer. If you want the region to apply to all of your destination buffers, you must do this yourself. For example, you can use QuickDraw's <code>OffsetRgn</code> function, which is described in the chapter "Basic QuickDraw" in <i>Inside Macintosh: Imaging</i> . If you do not want to specify a clipping region, set this field to <code>nil</code> .
list	Contains an array of output buffer specifications. Each buffer is represented by a buffer structure. The format and content of this structure are described in the next section.

The Buffer Structure

The `VdigBufferRec` data type defines a buffer structure.

```
typedef struct {
    PixMapHandle    dest;          /* handle to pixel map for
                                   destination buffer */
    Point           location;      /* location of video destination
                                   in pixel map */
    long            reserved;      /* reserved--set to 0 */
} VdigBufferRec;
```

Field descriptions

<code>dest</code>	Contains a handle to the pixel map that defines the destination buffer.
<code>location</code>	Specifies the location of the video destination in the pixel map specified by the <code>dest</code> field. This point identifies the upper-left corner of the destination rectangle. The size and scaling of the destination rectangle are governed by the <code>matrix</code> and <code>mask</code> fields of the buffer list structure that contains this structure.
<code>reserved</code>	Reserved for use by Apple. Set this field to 0.

Video Digitizer Component Functions

This section describes the functions that are provided by video digitizer components. These functions are described from the perspective of an application that uses video digitizer components. If you are developing a video digitizer component, your digitizer component must behave as described here.

This section has been divided into the following topics:

- “Getting Information About Video Digitizer Components” describes the functions that allow applications to obtain information about the capabilities of video digitizer components.
- “Setting Source Characteristics” discusses the video digitizer functions that allow applications to establish the source video environment.
- “Selecting an Input Source” describes how applications select the input video source.
- “Setting Video Destinations” describes the functions that allow applications to establish the destination display environment.
- “Controlling Compressed Source Devices” describes the functions that allow applications to work with devices that return compressed image data.
- “Controlling Digitization” describes functions that allow applications to start and stop digitization.
- “Controlling Color” discusses the functions that allow applications to control color mapping in the video digitizer component.

Video Digitizer Components

- “Controlling Analog Video” describes several functions that allow applications to control the characteristics of the input analog video signal.
- “Selectively Displaying Video” discusses functions that allow applications to work with the key colors that are used to control video display.
- “Clipping” discusses functions that allow applications to control the clipping region used by video digitizer components.
- “Utility Functions” describes a few utility functions that are supported by video digitizer components.

Note

If you are developing an application that uses video digitizer components, you should read the sections that are appropriate to your application. If you are developing a video digitizer component, you should read all the sections. ♦

These functions specify the video digitizer components for their requests with a reference obtained from the Component Manager’s `OpenComponent` function. See the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox* for details.

Getting Information About Video Digitizer Components

This section discusses functions that allow applications to obtain information about the capabilities and current state of video digitizer components.

You can use the `VDGetDigitizerInfo` function in your application to retrieve information about the capabilities of a video digitizer component. You can use the `VDGetCurrentFlags` function to obtain current status information from a video digitizer component.

VDGetDigitizerInfo

The `VDGetDigitizerInfo` function returns capability and status information about a specified video digitizer component.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetDigitizerInfo
                                (VideoDigitizerComponent ci,
                                 DigitizerInfo *info);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager’s <code>OpenComponent</code> function.
-----------------	---

Video Digitizer Components

info Contains a pointer to a digitizer information structure. The `VDGetDigitizerInfo` function returns information describing the capabilities of the specified video digitizer into this structure. See “The Digitizer Information Structure” on page 8-20 for a complete description.

DESCRIPTION

The `VDGetDigitizerInfo` function returns the capability and status information in a digitizer information structure (defined by the `DigitizerInfo` data type).

RESULT CODE

`noErr` 0 No error

SEE ALSO

Your application may also use the `VDGetCurrentFlags` function (described in the next section) to retrieve just the current status information about a video digitizer component.

VDGetCurrentFlags

The `VDGetCurrentFlags` function returns status information about a specified video digitizer component.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetCurrentFlags
                                (VideoDigitizerComponent ci,
                                 long *inputCurrentFlag,
                                 long *outputCurrentFlag);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager’s `OpenComponent` function.

inputCurrentFlag Contains a pointer to a long integer that is to receive the current input state flags for the video digitizer component. The `VDGetCurrentFlags` function returns the current input state flags into this location. See “Current Flags” on page 8-19 for a complete description of these flags.

outputCurrentFlag Contains a pointer to a long integer that is to receive the current output state flags for the video digitizer component. The `VDGetCurrentFlags` function returns the current output state flags into this location. See “Current Flags” on page 8-19 for a complete description of these flags.

DESCRIPTION

The `VDGetCurrentFlags` function returns the status information into two fields that contain flags specifying the current input and output status of the digitizer component.

You can also use the `VDGetDigitizerInfo` function (described in the previous section) in your application to retrieve capability and current status information about a video digitizer component.

The `VDGetCurrentFlags` function is often more convenient than the `VDGetDigitizerInfo` function. For example, this function provides a simple mechanism for determining whether a video digitizer is receiving a valid input signal. An application can retrieve the current input state flags and test the high-order bit by examining the sign of the returned value. If the value is negative (that is, the high-order bit, `digiInSignalLock`, is set to 1), the digitizer component is receiving a valid input signal.

RESULT CODE

`noErr` 0 No error

Setting Source Characteristics

This section discusses the video digitizer component functions that allow applications to set the spatial characteristics of the source video signal. You can use these functions in your application to set and retrieve information about the maximum source rectangle, the active source rectangle, the vertical blanking rectangle, and the digitizer rectangle. For a complete discussion of the relationship between these rectangles, see “About Video Digitizer Components,” which begins on page 8-3.

You can use the `VDGetMaxSrcRect` function in your application to get the size and location of the maximum source rectangle. Similarly, the `VDGetActiveSrcRect` function allows you to get this information about the active source rectangle, and the `VDGetVBlankRect` function enables you to obtain information about the vertical blanking rectangle.

You can use the `VDSetDigitizerRect` function to set the size and location of the digitizer rectangle. The `VDGetDigitizerRect` function lets you retrieve the size and location of this rectangle.

VDGetMaxSrcRect

The `VDGetMaxSrcRect` function returns the maximum source rectangle.

```
pascal VideoDigitizerError VDGetMaxSrcRect
                                (VideoDigitizerComponent ci,
                                 short inputStd,
                                 Rect *maxSrcRect);
```


Video Digitizer Components

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
inputStd	A short integer that specifies the input video signal associated with this maximum source rectangle.
maxSrcRect	Contains a pointer to a rectangle that is to receive the size and location information for the maximum source rectangle.

DESCRIPTION

The maximum source rectangle defines the spatial boundaries of the input video signal. All other rectangles—active source rectangle, digitizer rectangle, and vertical blanking rectangle—are defined relative to the maximum source rectangle. For a complete discussion of the relationship between these rectangles, see “About Video Digitizer Components,” which begins on page 8-3.

All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDGetActiveSrcRect

The `VDGetActiveSrcRect` function allows applications to obtain the size and location information for the active source rectangle used by a video digitizer component.

```
pascal VideoDigitizerError VDGetActiveSrcRect
                                (VideoDigitizerComponent ci,
                                 short inputStd,
                                 Rect *activeSrcRect);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
inputStd	A short integer that specifies the input video signal associated with this maximum source rectangle.
activeSrcRect	Contains a pointer to a rectangle that is to receive the size and location information for the active source rectangle.

DESCRIPTION

The source rectangle is that area in the source video image that contains active video. The video digitizer component returns spatial information that is relative to the maximum source rectangle. For a complete discussion of the relationship between these rectangles, see “About Video Digitizer Components,” which begins on page 8-3.

All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDGetVBlankRect

The `VDGetVBlankRect` function returns the vertical blanking rectangle.

```
pascal VideoDigitizerError VDGetVBlankRect
                                (VideoDigitizerComponent ci,
                                 short inputStd,
                                 Rect *vBlankRect);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

inputStd Specifies a short integer for the signaling standard used in the source video signal. Valid values are

`ntscIn` Input video signal to digitize is in NTSC format

`palIn` Input video signal to digitize is in PAL format

`secamIn` Input video signal to digitize is in SECAM format

vBlankRect

Contains a pointer to a rectangle that is to receive the size and location information for the vertical blanking rectangle.

DESCRIPTION

The vertical blanking rectangle defines the vertical blanking area in the input video signal, and it corresponds to lines 10 through 19 of the incoming video signal. The video digitizer component returns spatial information that is relative to the maximum source

rectangle. For a complete discussion of the relationship between these rectangles, see “About Video Digitizer Components,” which begins on page 8-3.

All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDSetDigitizerRect

The `VDSetDigitizerRect` function allows applications to set the current digitizer rectangle.

```
pascal VideoDigitizerError VDSetDigitizerRect
                                (VideoDigitizerComponent ci,
                                 Rect *digitizerRect);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
digitizerRect	Contains a pointer to a rectangle that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle.

DESCRIPTION

The current digitizer rectangle defines the area that the digitizer component reads from the input video signal. Applications can crop the input video signal by manipulating this rectangle. The digitizer rectangle coordinates must be specified relative to the maximum source rectangle. Furthermore, the digitizer rectangle must be completely within the maximum source rectangle. For a complete discussion of the relationship between these rectangles, see “About Video Digitizer Components,” which begins on page 8-3.

All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDGetDigitizerRect

The `VDGetDigitizerRect` function returns the current digitizer rectangle.

```
pascal VideoDigitizerError VDGetDigitizerRect
                                (VideoDigitizerComponent ci,
                                 Rect *digitizerRect);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

digitizerRect Contains a pointer to a rectangle that is to receive the size and location information for the current digitizer rectangle.

DESCRIPTION

The current digitizer rectangle defines the area that the digitizer component reads from the input video signal. The video digitizer component returns spatial information that is relative to the maximum source rectangle. For a complete discussion of the relationship between these rectangles, see "About Video Digitizer Components," which begins on page 8-3.

All video digitizer components must support this function.

RESULT CODE

`noErr` 0 No error

Selecting an Input Source

This section discusses the video digitizer component functions that allow applications to select an input video source.

Some of these functions provide information about the available video inputs. Applications can use the `VDGetNumberOfInputs` function to determine the number of video inputs supported by the digitizer component. The `VDGetInputFormat` function allows applications to find out the video format (composite, s-video, or component) employed by a specified input.

You can use the `VDSetInput` function in your application to specify the input to be used by the digitizer component. The `VDGetInput` function returns the currently selected input.

The `VDSetInputStandard` function allows you to specify the video signaling standard to be used by the video digitizer component.

VDGetNumberOfInputs

The `VDGetNumberOfInputs` function returns the number of input video sources that a video digitizer component supports.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetNumberOfInputs
                                   (VideoDigitizerComponent ci,
                                   short *inputs);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>inputs</code>	Contains a pointer to an integer that is to receive the number of input video sources supported by the specified component. Video digitizer components number video sources sequentially, starting at 0. So, if a digitizer component supports two inputs, this function sets the field referred to by the <code>inputs</code> parameter to 1.

RESULT CODE

`noErr` 0 No error

VDSetInput

The `VDSetInput` function allows applications to select the input video source for a video digitizer component.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDSetInput (VideoDigitizerComponent ci,
                                         short input);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>input</code>	Specifies the input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request the first video source, an application sets this parameter to 0.

Video Digitizer Components

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

SEE ALSO

Applications can get the number of video sources supported by a video digitizer component by calling the `VDGetNumberOfInputs` function (described in the previous section). Applications can get more information about a video source by calling the `VDGetInputFormat` function (described on page 8-32).

VDGetInput

The `VDGetInput` function returns data that identifies the currently active input video source.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetInput (VideoDigitizerComponent ci,
                                       short *input);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
input	Contains a pointer to a short integer that is to receive the identifier for the currently active input video source. Video digitizer components number video sources sequentially, starting at 0. So, if the first source is active, this function sets the field referred to by the <code>input</code> parameter to 0.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDGetInputFormat

The `VDGetInputFormat` function allows applications to determine the format of the video signal provided by a specified video input source.

```
pascal VideoDigitizerError VDGetInputFormat
                               (VideoDigitizerComponent ci,
                                short input, short *format);
```

Video Digitizer Components

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
input	Specifies the input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling the <code>VDGetNumberOfInputs</code> function, discussed on page 8-31.
format	Contains a pointer to a short integer that is to receive the specification of the video format of the specified input source. This function updates the field referred to by the <code>format</code> parameter. Valid values are <code>compositeIn</code> The input video signal is in composite format <code>sVideoIn</code> The input video signal is in s-video format <code>rgbComponentIn</code> The input video signal is in RGB component format

DESCRIPTION

Video digitizer components support three video formats: composite video, s-video, and component video (RGB signal).
All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDSetInputStandard

The `VDSetInputStandard` function allows applications to specify the input signaling standard to digitize. Video digitizer components support three input signaling standards: NTSC, PAL, and SECAM.

```
pascal VideoDigitizerError VDSetInputStandard
                                (VideoDigitizerComponent ci,
                                 short inputStandard);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
----	---

Video Digitizer Components

`inputStandard`

A short integer that specifies the signaling standard used in the source video signal. Valid values are

<code>ntscIn</code>	Input video signal to digitize is in NTSC format
<code>palIn</code>	Input video signal to digitize is in PAL format
<code>secamIn</code>	Input video signal to digitize is in SECAM format

DESCRIPTION

Applications can use the `VDGetDigitizerInfo` function (described on page 8-24) to determine the capabilities of a specified video digitizer component. Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to determine the current input state of a digitizer component.

All video digitizer components must support this function.

SPECIAL CONSIDERATIONS

Your digitizer component should ensure that spatial characteristics that were set for one standard are not interpreted within another standard.

RESULT CODES

<code>noErr</code>	0	No error
<code>qtParamErr</code>	-2202	Invalid parameter value

Setting Video Destinations

Video digitizer components provide several functions that allow applications to specify the destination for the digitized video stream produced by the digitizer component. Applications have two options for specifying the destination for the video data stream.

The first option requires that the video be digitized as RGB pixels and placed into a destination pixel map. This option allows the video to be placed either onscreen or offscreen, depending upon the placement of the pixel map. You can use the `VDSetPlayThruDestination` function in your application to set the characteristics for this option. The `VDPreFlightDestination` function lets you determine the capabilities of the digitizer in your application. All video digitizer components must support this option. The `VDGetPlayThruDestination` function lets you get data about the current video destination.

The second option uses a global boundary rectangle to define the destination for the video. This option is useful only with digitizers that support hardware DMA. You can use the `VDSetPlayThruGlobalRect` function in your application to set the characteristics for this option. You can use the `VDPreFlightGlobalRect` function in your application to determine the capabilities of the digitizer. Not all video digitizer components support this option.

The `VDGetMaxAuxBuffer` function returns information about a buffer that may be located on some special hardware.

VDSetPlayThruDestination

You can use the `VDSetPlayThruDestination` function in your application to establish the destination settings for a video digitizer component.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDSetPlayThruDestination
                                (VideoDigitizerComponent ci,
                                 PixMapHandle dest,
                                 Rect *destRect,
                                 MatrixRecord *m, RgnHandle mask);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

dest Contains a handle to the destination pixel map. This pixel map may be in the video frame buffer of the Macintosh computer, or it may specify an offscreen buffer.

The video digitizer component examines this pixel map to determine the display characteristics of the video destination, including the base address, row bytes, and pixel depth. If the digitizer component does not support these characteristics, it sets the return value to `badDepth`. If the digitizer component cannot accommodate the location of the destination pixel map, it sets the return value to `noDMA`.

If you are going to use multiple output buffers, be sure to include this buffer in the buffer list that you define with the `VDSetupBuffers` function, which is described on page 8-54. You may call the `VDSetupBuffers` function before calling `VDSetPlayThruDestination`.

destRect Contains a pointer to a rectangle that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination pixel map specified by the `dest` parameter.

This is an optional parameter. Applications may specify a transformation matrix to control the placement and scaling of the video image in the destination pixel map. In this case, the `destRect` parameter is set to `nil` and the `m` parameter specifies the matrix.

If the `destRect` parameter is `nil`, you can determine the destination rectangle for simple matrices by calling the `TransformRect` function using the current digitizer rectangle and this matrix. For more information on `TransformRect`, see the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime*.

Video Digitizer Components

<code>m</code>	<p>Contains a pointer to a matrix structure containing the transformation matrix for the destination video image. To determine the capabilities of a video digitizer component, you can call the <code>VDGetDigitizerInfo</code> function, described on page 8-24, in your application.</p> <p>This is an optional parameter. Applications may specify a destination rectangle to control the placement and scaling of the video image in the destination pixel map. In this case, the <code>m</code> parameter is set to <code>nil</code> and the <code>destRect</code> parameter specifies the destination rectangle.</p>
<code>mask</code>	<p>Contains a region handle that defines a mask. Applications can use masks to control clipping of the video into the destination rectangle. This mask region is defined in the destination coordinate space.</p> <p>This is an optional parameter. Applications may use alpha channels or key colors to control video blending. If there is no mask, applications should set the <code>mask</code> parameter to <code>nil</code>.</p>

DESCRIPTION

The application provides the desired settings as parameters to this function. Applications should verify that the video digitizer component can accommodate the settings by calling the `VDPreflightDestination` function, described in the next section.

Applications set the source digitizer rectangle by calling the `VDSetDigitizerRect` function, described on page 8-29.

RESULT CODES

<code>noErr</code>	0	No error
<code>badDepth</code>	-2207	Digitizer cannot accommodate pixel depth
<code>noDMA</code>	-2208	Digitizer cannot use DMA to this destination

VDPreflightDestination

You can use the `VDPreflightDestination` function in your application to verify that a video digitizer component can support a set of destination settings intended for use with the `VDSetPlayThruDestination` function, which is described in the previous section.

```
pascal VideoDigitizerError VDPreflightDestination
                                (VideoDigitizerComponent ci,
                                 Rect *digitizerRect,
                                 PixMapHandle dest,
                                 Rect *destRect,
                                 MatrixRecord *m);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>digitizerRect</code>	<p>Contains a pointer to a rectangle that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle. For a complete discussion of the relationship between these rectangles, see "About Video Digitizer Components," which begins on page 8-3.</p> <p>If the video digitizer component cannot accommodate the specified rectangle, it changes the coordinates in this structure to specify a rectangle that it can support and sets the result to <code>qtParamErr</code>.</p>
<code>dest</code>	Contains a handle to the destination pixel map.
<code>destRect</code>	<p>Contains a pointer to a rectangle that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination pixel map specified by the <code>dest</code> parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to <code>qtParamErr</code>.</p> <p>This is an optional parameter. Applications may specify a transformation matrix to control the placement and scaling of the video image in the destination pixel map. In this case, the <code>destRect</code> parameter is set to <code>nil</code> and the <code>m</code> parameter specifies the matrix.</p>
<code>m</code>	<p>Contains a pointer to a matrix structure containing the transformation matrix for the destination video image. If the video digitizer component cannot accommodate this matrix, it changes the values in the structure to define a matrix that it can support and sets the result to <code>qtParamErr</code>. Applications can determine the capabilities of a video digitizer component by calling the <code>VDGetDigitizerInfo</code> function, described on page 8-24.</p> <p>This is an optional parameter. Applications may specify a destination rectangle to control the placement and scaling of the video image in the destination pixel map. In this case, the <code>m</code> parameter is set to <code>nil</code> and the <code>destRect</code> parameter specifies the destination rectangle.</p> <p>If the <code>destRect</code> parameter is <code>nil</code>, you can determine the destination rectangle for simple matrices by calling the <code>TransformRect</code> function using the current digitizer rectangle and this matrix. For more information on <code>TransformRect</code>, see the chapter "Movie Toolbox" in <i>Inside Macintosh: QuickTime</i>.</p>

DESCRIPTION

The application provides the desired settings as parameters to this function. The video digitizer component then examines those settings. If the digitizer component can support the specified settings, it sets the result code to `noErr`. If the digitizer component cannot support the settings, it alters the input settings to reflect values that it can support and returns a result code of `qtParamErr`. The application can then use the settings with the `VDSetPlayThruDestination` function (described in the previous section).

All video digitizer components must support this function.

Applications should use the `VDPreFlightDestination` function to test destination settings whenever the video digitizer component cannot support arbitrary scaling.

RESULT CODES

<code>noErr</code>	0	No error
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Applications can determine the capabilities of a video digitizer component by examining the output capability flags (see the discussion of the `VDGetCurrentFlags` function, which begins on page 8-25, for more information about retrieving these flags).

Specifically, if the `digiOutDoesStretch` and `digiOutDoesShrink` flags are set to 1 in the output capability flag, the digitizer component supports arbitrary scaling.

VDGetPlayThruDestination

The `VDGetPlayThruDestination` function allows applications to obtain information about the current video destination.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetPlayThruDestination
                                (VideoDigitizerComponent ci,
                                 PixMapHandle *dest, Rect *destRect,
                                 MatrixRecord *m, RgnHandle *mask);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>dest</code>	Contains a pointer to a pixel map handle. The video digitizer component returns a handle to the destination pixel map in the field referred to by this parameter. It is the caller's responsibility to dispose of the pixel map.

Video Digitizer Components

<code>destRect</code>	Contains a pointer to a rectangle structure. The video digitizer component places the coordinates of the output rectangle into the structure referred to by this parameter. If there is no output rectangle defined, the component returns an empty rectangle.
<code>m</code>	Contains a pointer to a matrix structure. The video digitizer component places the transformation matrix into the structure referred to by this parameter.
<code>mask</code>	Contains a pointer to a region handle. The video digitizer component places a handle to the mask region into the field referred to by this parameter. Applications can use masks to control the video into the destination rectangle. For more information about masks, see “About Video Digitizer Components,” which begins on page 8-3. If there is no mask region defined, the digitizer component sets this returned handle to <code>nil</code> . The caller is responsible for disposing of this region.

DESCRIPTION

Applications can set the video destination by calling either the `VDSetPlayThruDestination` function (described on page 8-35) or the `VDSetPlayThruGlobalRect` function (described in the next section). Applications should call the `VDGetPlayThruDestination` function only after having set the destination with the `VDSetPlayThruDestination` function.

RESULT CODE

`noErr` 0 No error

VDSetPlayThruGlobalRect

You can use the `VDSetPlayThruGlobalRect` function in your application to establish the destination settings for a video digitizer component that is to digitize into a global rectangle. The application provides the desired settings as parameters to this function. Not all video digitizer components support global rectangles.

```
pascal VideoDigitizerError VDSetPlayThruGlobalRect
                                (VideoDigitizerComponent ci,
                                 GrafPtr theWindow,
                                 Rect *globalRect);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>theWindow</code>	Contains a pointer to the destination window.

Video Digitizer Components

`globalRect`

Contains a pointer to a rectangle that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the `theWindow` parameter.

DESCRIPTION

Applications should verify that the digitizer component can accommodate the settings by calling the `VDPreFlightGlobalRect` function, described in the next section.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Applications set the source digitizer rectangle by calling the `VDSetDigitizerRect` function, described on page 8-29.

VDPreFlightGlobalRect

You can use the `VDPreFlightGlobalRect` function in your application to verify that a video digitizer component can support a set of destination settings intended for use with the `VDSetPlayThruGlobalRect` function (described in the previous section).

```
pascal VideoDigitizerError VDPreflightGlobalRect
                                (VideoDigitizerComponent ci,
                                 GrafPtr theWindow,
                                 Rect *globalRect);
```

`ci` Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

`theWindow` Contains a pointer to the destination window.

`globalRect` Contains a pointer to a rectangle that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the `theWindow` parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

DESCRIPTION

The application provides the desired settings as parameters to this function. The video digitizer component then examines those settings. If the digitizer component can support the specified settings, it sets the result code to `noErr`. If the digitizer component cannot support the settings, it alters the input settings to reflect values that it can support and returns a result code of `qtParamErr`.

Applications should use this function to determine whether a video digitizer supports placing destination video into a rectangle that crosses screens. Digitizers that do not support this capability return a result of `digiUnimpErr`.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

VDGetMaxAuxBuffer

The `VDGetMaxAuxBuffer` function allows applications to obtain access to buffers that are located on special hardware. Digitizer components that are constrained to a single output device can provide an auxiliary buffer to support multiple buffering.

```
pascal VideoDigitizerError VDGetMaxAuxBuffer
                                (VideoDigitizerComponent ci,
                                 PixMapHandle *pm, Rect *r);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>pm</code>	Contains a pointer to a pixel map handle. The video digitizer component returns a handle to the destination pixel map in the field referred to by this parameter. Do not dispose of this pixel map. If the digitizer component cannot allocate a buffer, this handle is set to <code>nil</code> .
<code>r</code>	Contains a pointer to a rectangle structure. The video digitizer component places the coordinates of the largest output rectangle it can support into the structure referred to by this parameter.

DESCRIPTION

You can use the `VDGetMaxAuxBuffer` function in your application to determine whether a video digitizer component supports an auxiliary buffer. If the digitizer component provides an auxiliary buffer, it is to your advantage to use it. By using the buffer, you may achieve better performance under some circumstances, such as when the digitizer component does not support DMA.

RESULT CODES

Controlling Compressed Source Devices

Some video digitizer components may provide functions that allow applications to work with digitizing devices that can provide compressed image data directly. Such devices allow applications to retrieve compressed image data without using the Image Compression Manager. However, in order to display images from the compressed data stream, there must be an appropriate decompressor component available to decompress the image data.

Video digitizers that can support compressed source devices set the `digiOutDoesCompress` flag to 1 in their capability flags (see “Capability Flags” beginning on page 8-14 for more information about these flags).

Applications can use the `VDGetCompressionTypes` function to determine the image-compression capabilities of a video digitizer. The `VDSetCompression` function allows applications to set some parameters that govern image compression.

Applications control digitization by calling the `VDCompressOneFrameAsync` function, which instructs the video digitizer to create one frame of compressed image data. The `VDCompressDone` function returns that frame. When an application is done with a frame, it calls the `VDReleaseCompressBuffer` function to free the buffer. An application can force the digitizer to place a key frame into the sequence by calling the `VDResetCompressSequence` function. Applications can turn compression on and off by calling `VDSetCompressionOnOff`.

Applications can obtain the digitizer’s image description structure by calling the `VDGetImageDescription` function. Applications can set the digitizer’s time base by calling the `VDSetTimeBase` function.

All of the digitizing functions described in this section support only asynchronous digitization. That is, the video digitizer works independently to digitize each frame. Applications are free to perform other work while the digitizer works on each frame.

The video digitizer component manages its own buffer pool for use with these functions. In this respect, these functions differ from the other video digitizer functions that support asynchronous digitization (see “Controlling Digitization” beginning on page 8-52 for more information about these functions).

VDGetCompressionTypes

The `VDGetCompressionTypes` function allows an application to determine the image-compression capabilities of the video digitizer.

```
pascal VideoDigitizerError VDGetCompressionTypes
                                (VideoDigitizerComponent ci,
                                 VDCompressionListHandle h);
```


Video Digitizer Components

ci	Identifies an application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
h	Identifies a handle to receive the compression information. The video digitizer returns information about its capabilities by formatting one or more compression list structures in this handle (the format and content of the compression list structure are discussed later). If the digitizer supports more than one compression type, it creates an array of structures in this handle. The video digitizer sizes this handle appropriately. It is the application's responsibility to dispose of this handle when it is done with it.

DESCRIPTION

The video digitizer places its preferred, or default, compression options in the first compression list structure in the returned array.

Note that there must be a decompressor component of the appropriate type available in the system if an application is to display images from a compressed image sequence.

The `VDCompressionList` data type defines the format and content of the compression list structure:

```
typedef struct VDCompressionList {
    CodecComponent    codec;           /* component ID */
    CodecType         cType;           /* compressor type */
    Str63             typeName;        /* compression algorithm */
    Str63             name;            /* compressor name string */
    long              formatFlags;     /* data format flags */
    long              compressFlags;   /* capabilities flags */
    long              reserved;        /* set to 0 */
} VDCompressionList, *VDCompressionListPtr,
**VDCompressionListHandle;
```

Field descriptions

codec	Contains the component identifier for the video digitizer's compressor component. Some video digitizers may also implement their image-compression capabilities in an Image Compression Manager compressor component. In this case, the digitizer may allow the application to connect to and use the compressor. If so, the digitizer provides the compressor component's identifier here. If not, the digitizer sets this field to <code>nil</code> .
cType	Identifies the compression algorithm supported by the video digitizer. See the chapter "Image Compression Manager" in <i>Inside Macintosh: QuickTime</i> for a list of values supported by Apple.
typeName	Contains a text string that identifies the compression algorithm. An application may display this string to the user to identify the type of image compression being performed. See the chapter "Image

Video Digitizer Components

	Compression Manager” in <i>Inside Macintosh: QuickTime</i> for a list of values supported by Apple.
name	Specifies the name of the compressor. The developer of the video digitizer assigns this name. An application may display this string to the user.
formatFlags	Contains flags that describe the data formats supported by the video digitizer. Typically, these flags are of interest only to developers of video digitizers and image compressors. See the chapter “Image Compressor Components” in this book for more information.
compressFlags	Contains flags that describe the compression capabilities of the video digitizer. Typically, these flags are of interest only to developers of video digitizers and image compressors. See the chapter “Image Compressor Components” in this book for more information.
reserved	Reserved for Apple. Always set to 0.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

VDSetCompression

The `VDSetCompression` function allows applications to specify some compression parameters.

```
pascal VideoDigitizerError VDSetCompression
                                (VideoDigitizerComponent ci,
                                 OSType compressType, short depth,
                                 Rect *bounds, CodecQ spatialQuality,
                                 CodecQ temporalQuality,
                                 long keyFrameRate);
```

ci Identifies the application’s connection to the video digitizer component. An application obtains this value from the Component Manager’s `OpenComponent` function.

compressType Specifies a compressor type. This value corresponds to the component subtype of the compressor component. See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* for more information about compressor types and for valid values for this parameter.

Video Digitizer Components

depth	Specifies the depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 correspond to 1-bit, 2-bit, 4-bit, and 8-bit grayscale images.
bounds	Contains a pointer to a rectangle that defines the desired boundaries of the compressed image.
spatialQuality	Indicates the desired image quality for each frame in the sequence. See the chapter “Image Compression Manager” in <i>Inside Macintosh: QuickTime</i> for valid compression quality values.
temporalQuality	Indicates the desired temporal quality for the sequence as a whole. See the chapter “Image Compression Manager” in <i>Inside Macintosh: QuickTime</i> for valid compression quality values.
keyFrameRate	Specifies the maximum number of frames to allow between key frames. This value defines the minimum rate at which key frames are to appear in the compressed sequence; however, the video digitizer may insert key frames more often than an application specifies. If the application requests no temporal compression (that is, the application set the <code>temporalQuality</code> parameter to 0), the video digitizer ignores this parameter. For more information about key frames, see the chapter “Image Compression Manager” in <i>Inside Macintosh: QuickTime</i> .

DESCRIPTION

An application may use the `VDSetCompression` function to control the parameters that govern image compression. An application may change the compressor type, image depth, and boundary rectangle parameters only when the digitizer is stopped. However, if an application sets these three parameters (that is, the `compressType`, `depth`, and `bounds` parameters) to 0, it may work with the other parameters while digitization is active. This allows an application to vary the data rate during digitization.

RESULT CODES

<code>noErr</code>	<code>0</code>	No error
<code>digiUnimpErr</code>	<code>-2201</code>	Function not supported
<code>qtParamErr</code>	<code>-2202</code>	Invalid parameter value

VDSetCompressionOnOff

The `VDSetCompressionOnOff` function allows an application to start and stop compression by digitizers that can deliver either compressed or uncompressed image data.

```
pascal VideoDigitizerError VDSetCompressionOnOff
                                (VideoDigitizerComponent ci,
                                 Boolean state);
```

<code>ci</code>	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
<code>state</code>	Contains a Boolean value that indicates whether to enable or disable compression. Applications set this parameter to <code>true</code> to enable compression. Setting it to <code>false</code> disables compression.

DESCRIPTION

This is a required function for digitizers that are going to perform compression. These digitizers have their `digiOutDoesCompress` capability flag set to 1 and their `digiOutDoesCompressOnly` flag set to 0. Digitizers that support this capability typically deliver uncompressed image data in addition to the compressed data stream; the uncompressed data is ready for display.

Digitizers that only provide compressed data have their `digiOutDoesCompressOnly` flag set to 1, rather than 0. These digitizers may either ignore this function or return a nonzero result code.

Applications must call this function before they call either `VDSetCompression` or `VDCompressOneFrameAsync`. This allows the video digitizer to prepare for the operation.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDCompressOneFrameAsync

The `VDCompressOneFrameAsync` function instructs the video digitizer to digitize and compress a single frame of image data. Because the component performs this action asynchronously, the application is free to do other things while the digitizer works on the image.

```
pascal VideoDigitizerError VDCompressOneFrameAsync
                               (VideoDigitizerComponent ci);
```

`ci` Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

DESCRIPTION

An application can determine when the digitizer is done with the frame by calling the `VDCompressDone` function, which is discussed next.

Unlike the `VDGrabOneFrameAsync` function (discussed on page 8-56), the video digitizer handles all details of managing data buffers.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDCompressDone

The `VDCompressDone` function allows an application to determine whether the video digitizer has finished digitizing and compressing a frame of image data. An application starts the digitizing process by calling the `VDCompressOneFrameAsync` function, which was just discussed.

```
pascal VideoDigitizerError VDCompressDone
                                (VideoDigitizerComponent ci,
                                 Boolean *done, Ptr *theData,
                                 long *dataSize,
                                 unsigned char *similarity,
                                 TimeRecord *t);
```

<code>ci</code>	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
<code>done</code>	Contains a pointer to a Boolean value. Applications set this value to <code>true</code> when they are done, and set it to <code>false</code> if the operation is incomplete.
<code>theData</code>	Contains a pointer to a field that is to receive a pointer to the compressed image data. The digitizer returns a pointer that is valid in the application's current memory mode. The digitizer allocates the memory into which it places the digitized data. An application must call the <code>VDReleaseCompressBuffer</code> function to dispose of this memory; this function is discussed next.
<code>dataSize</code>	Contains a pointer to a field to receive a value indicating the number of bytes of compressed image data.
<code>similarity</code>	Contains a pointer to a field to receive an indication of the relative similarity of this image to the previous image in a sequence. A value of 0 indicates that the current frame is a key frame in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254). This field is only filled in if the temporal quality passed in with the <code>VDSetCompression</code> function (described on page 8-44) is not 0—that is, if it is not frame-differenced.
<code>t</code>	Contains a pointer to a time record. When the operation is complete, the digitizer fills in this structure with information indicating when the frame was grabbed. The time value stored in this structure is in the time base that the application sets with the <code>VDSetTimeBase</code> function (see page 8-51 for more information about this function). The format and content of this structure are discussed in the chapter “Movie Toolbox” in <i>Inside Macintosh: QuickTime</i> .

DESCRIPTION

An application can determine when the digitizer is done with the frame by calling the `VDCompressDone` function. When the digitizer is done, it sets the Boolean value referred to by the `done` parameter to `true`, and then returns information about the digitized and compressed frames via the `theData`, `dataSize`, `similarity`, and `t` parameters.

If the digitizer is not yet done, it sets the Boolean value to `false`. In this case, the digitizer does not return any other information.

Note that the digitizer is careful to return the frames in temporal order, and to avoid returning two frames with the same time value (unless the rate is set to 0).

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Applications must use the `VDReleaseCompressBuffer` function to free the memory that contains the compressed image data. This function is described in the next section.

VDReleaseCompressBuffer

The `VDReleaseCompressBuffer` function allows an application to free a buffer received from the `VDCompressDone` function.

```
pascal VideoDigitizerError VDReleaseCompressBuffer
                                (VideoDigitizerComponent ci,
                                 Ptr bufferAddr);
```

`ci` Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

`bufferAddr` Points to the location of the buffer to be released. This address must correspond to a buffer address that the application obtained from the `VDCompressDone` function (discussed in the previous section).

DESCRIPTION

Once an application frees the buffer, the video digitizer is able to use the buffer for other images. Applications should try to free these buffers as quickly as possible, so that the video digitizer can make optimum use of its buffer, and thereby support higher frame rates.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

VDGetImageDescription

The `VDGetImageDescription` function allows an application to retrieve an image description structure from a video digitizer.

```
pascal VideoDigitizerError VDGetImageDescription
                                (VideoDigitizerComponent ci,
                                 ImageDescriptionHandle desc);
```

ci	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
desc	Specifies a handle. The video digitizer fills this handle with an Image Compression Manager image description structure containing information about the digitizer's current compression settings. The digitizer resizes the handle appropriately. It is the application's responsibility to dispose of this handle.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

SEE ALSO

See the chapter "Image Compression Manager" in *Inside Macintosh: QuickTime* for a complete description of the image description structure.

VDRestCompressSequence

The `VDRestCompressSequence` function allows an application to force the video digitizer to insert a key frame into a temporally compressed image sequence.

```
pascal VideoDigitizerError VDRestCompressSequence
                                (VideoDigitizerComponent ci);
```

ci Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

DESCRIPTION

After an application calls this function, the digitizer ensures that the next frame returned to the application is a key frame.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

An application can control the rate at which the digitizer inserts key frames by calling the `VDSetCompression` function, which is discussed beginning on page 8-44.

VDSetTimeBase

The `VDSetTimeBase` function allows an application to establish the video digitizer's time coordinate system.

```
pascal VideoDigitizerError VDSetTimeBase
                                (VideoDigitizerComponent ci,
                                TimeBase t);
```

ci Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

t Specifies the video digitizer's new time base.

DESCRIPTION

Video digitizers return all time information in relation to the specified time base. For example, whenever a digitizer returns a compressed frame from its `VDCompressDone` function, it returns time information relating to the time when the frame was digitized and compressed. This time information is expressed in the time base that the application specifies with this function.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

Controlling Digitization

This section describes the video digitizer component functions that allow applications to control video digitization. Video digitizer components allow applications to start and stop the digitizing process. Your application can request continuous digitization or single-frame digitization. When a digitizer component is operating continuously, it automatically places successive frames of digitized video into the specified destination. When a digitizer component works with a single frame at a time, the application and other software, such as an image compressor component, control the speed at which the digitized video is processed.

You can use the `VDSetPlayThruOnOff` function in your application to enable or disable digitization. When digitization is enabled, the video digitizer component places digitized video frame into the specified destination continuously. The application stops the digitizer by disabling digitization. This function can be used with both destination options.

Alternatively, your application can control digitization on a frame-by-frame basis. The `VDGrabOneFrame` and `VDGrabOneFrameAsync` functions digitize a single video frame; `VDGrabOneFrame` works synchronously, returning control to your application when it has obtained a complete frame, while `VDGrabOneFrameAsync` works asynchronously. The `VDDone` function helps you to determine when the `VDGrabOneFrameAsync` function is finished with a video frame. Your application can define the buffers for use with asynchronous digitization by calling the `VDSetupBuffers` function. Free the buffers by calling the `VDReleaseAsyncBuffers` function.

The `VDSetFrameRate` function allows applications to control the digitizer's frame rate. The `VDGetDataRate` function returns the digitizer's current data rate.

VDSetPlayThruOnOff

The `VDSetPlayThruOnOff` function allows applications to control continuous digitization.

```
pascal VideoDigitizerError VDSetPlayThruOnOff
                                (VideoDigitizerComponent ci,
                                 short state);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>state</code>	<p>A short integer that specifies whether to use continuous digitization. The following values are valid:</p> <p><code>digitizerOff</code> Turns off continuous digitization</p> <p><code>digitizerOn</code> Turns on continuous digitization</p> <p>When an application stops continuous digitization, the video digitizer component must restore its alpha channel, blending mask, or key color settings to graphics mode.</p>

DESCRIPTION

When opened, video digitizer components are always set to off, so that no digitization is taking place. Your application can use the `VDSetPlayThruOnOff` function to turn continuous digitization on and off.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Applications can also use single-frame digitization by calling the `VDGrabOneFrame` or `VDGrabOneFrameAsync` function, described in the next section and on page 8-56, respectively.

VDGrabOneFrame

The `VDGrabOneFrame` function instructs the video digitizer component to digitize a single frame of source video.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGrabOneFrame
                                (VideoDigitizerComponent ci);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

DESCRIPTION

The application specifies the destination for the digitized frame by calling either the `VDSetPlayThruDestination` function (described on page 8-35) or the `VDSetPlayThruGlobalRect` function (described on page 8-39).

If the specified digitizer component is already digitizing continuously when the application calls `VDGrabOneFrame`, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call the `VDSetPlayThruOnOff` function, which is described in the previous section.

The `VDGrabOneFrame` function supports synchronous single-frame video digitization—that is, the digitizer component does not return control to your application until it has successfully processed the next video frame. Some video digitizer components may also support asynchronous single-frame digitization. Applications can use asynchronous digitization by calling the `VDGrabOneFrameAsync` function, described on page 8-56.

RESULT CODE

`noErr` 0 No error

VDSetupBuffers

The `VDSetupBuffers` function allows applications to define output buffers for use with asynchronous grabs. Video digitizer components extract information about the spatial characteristics of the video destinations from these buffers.

```
pascal VideoDigitizerError VDSetupBuffers
                                (VideoDigitizerComponent ci,
                                VdigBufferRecListHandle bufferList);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>bufferList</code>	Contains a handle to a list of output buffers. This buffer list is contained in a buffer list structure. This structure is described in "The Buffer List Structure" on page 8-22. Note that the video digitizer component makes a copy of the buffer list—you may dispose of this handle when the function returns to your application.

▲ **WARNING**
If you are developing a video digitizer component, note that the `matrix` field in the buffer list structure contains a pointer to the matrix structure. It is your responsibility to copy that matrix structure. ▲

SPECIAL CONSIDERATIONS

Applications must define the output buffers before starting an asynchronous grab.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value
<code>badDepth</code>	-2207	Digitizer cannot accommodate specified depth
<code>noDMA</code>	-2208	Digitizer cannot use DMA to this destination

SEE ALSO

Applications instruct digitizer components to grab a single frame by calling the `VDGrabOneFrameAsync` function, which is described on page 8-56.

Applications free these buffers by calling the `VDReleaseAsyncBuffers` function, which is described next.

VDReleaseAsyncBuffers

The `VDReleaseAsyncBuffers` function allows an application to release the buffers that it allocates with the `VDSetupBuffers` function.

```
pascal VideoDigitizerError VDReleaseAsyncBuffers
                                (VideoDigitizerComponent ci);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
-----------------	---

DESCRIPTION

Applications release the buffers used in an asynchronous grab by calling the `VDReleaseAsyncBuffers` function.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Applications allocate buffers for asynchronous grabs by calling the `VDSetupBuffers` function, which is discussed in the previous section.

VDGrabOneFrameAsync

The `VDGrabOneFrameAsync` function instructs the video digitizer component to start to digitize asynchronously a single frame of source video. Because the component digitizes the video asynchronously, the application is free to do other things while the digitization is performed.

```
pascal VideoDigitizerError VDGrabOneFrameAsync
                                (VideoDigitizerComponent ci,
                                 short buffer);
```

`ci` Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

`buffer` Identifies the next output buffer. The value of this parameter must correspond to a valid index into the list of buffers that you supply when your application calls the `VDSetupBuffers` function (which is described on page 8-54). Note that this value is zero-based (that is, you must set this parameter to 0 to refer to the first buffer in the buffer list).

The video digitizer component uses this buffer for the *next* video frame (that is, the frame that will be digitized the next time the application calls the `VDGrabOneFrameAsync` function). In this manner, video digitizer components can quickly and efficiently prepare for the next video frame.

Some digitizer components may not allow your application to queue more than one asynchronous frame grab at a time. These components may not return control to your application until a previously requested grab has been completed.

DESCRIPTION

Applications determine when the digitizer component is finished with a frame by calling the `VDDone` function, which is described in the next section.

When calling the `VDGrabOneFrameAsync` function, the application specifies the next destination video buffer, allowing the digitizer component to quickly switch from the current buffer to the next buffer. In this manner, your application's ability to grab video at high frame rates is enhanced. See "Multiple Buffering" on page 8-8 for a discussion of multiple-buffered video digitization.

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by examining the output capability flags of the digitizer component. Specifically, if the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports the `VDGrabOneFrameAsync` function and the `VDDone` function, which is described in the next section.

Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to retrieve the digitizer component's output capability flags. If a video digitizer component does not support asynchronous digitization, applications must use the `VDGrabOneFrame` function (described on page 8-54) to perform single-frame digitization.

If the specified digitizer component is already digitizing continuously when the application calls `VDGrabOneFrameAsync`, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call the `VDSetPlayThruOnOff` function, which is described on page 8-53.

The `VDGrabOneFrameAsync` function also allows applications to use more than one destination buffer for the digitized video. The application defines these buffers by calling the `VDSetupBuffers` function (described on page 8-54). The application specifies one of these destination buffers for the digitized frame when it calls the `VDSetPlayThruDestination` function (described on page 8-35) or the `VDSetPlayThruGlobalRect` function (described on page 8-39).

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDDone

You can use the `VDDone` function in your application to determine if the `VDGrabOneFrameAsync` function is finished with a specific output buffer (`VDGrabOneFrameAsync` is described in the previous section). Applications that use the `VDGrabOneFrameAsync` function to digitize video frames should call `VDDone` before working with a digitized image.

```
pascal long VDDone (VideoDigitizerComponent ci, short buffer);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>buffer</code>	Identifies the buffer for the operation. The value of this parameter must correspond to a valid index into the list of buffers you supply when your application calls the <code>VDSetupBuffers</code> function (which is described on page 8-54). Note that this value is zero-based (that is, you must set this parameter to 0 to refer to the first buffer in the buffer list).

DESCRIPTION

If the `VDDone` function returns a 0 result, the video digitizer component has not finished the specified asynchronous frame grab. If the result is nonzero, the frame has been processed and the application can proceed to use the contents of the specified buffer.

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by examining the output capability flags of the digitizer component. Specifically, if the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports the `VDGrabOneFrameAsync` and `VDDone` functions. Applications can use the `VDGetCurrentFlags` function to retrieve the component's output capability flags. See page 8-25 for a description of the `VDGetCurrentFlags` function.

The `VDDone` function returns a long integer indicating whether the specified asynchronous frame grab is complete. If the returned value is 0, the video digitizer component is still working on the frame. If the returned value is nonzero, the digitizer component is finished with the frame and the application can perform its processing.

VDSetFrameRate

The `VDSetFrameRate` function allows an application to indicate its desired frame rate to the video digitizer. Note that some digitizers may not be able to support high frame rates.

```
pascal VideoDigitizerError VDSetFrameRate
                                   (VideoDigitizerComponent ci,
                                   Fixed framesPerSecond);
```

`ci` Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

`framesPerSecond` Specifies the application's desired frame rate. Applications may set this parameter to 0 to return the digitizer to its default frame rate (typically 29.97 frames per second).

DESCRIPTION

In some cases, the digitizer component may not be able to control its frame rate. These digitizers can run at only a single rate of speed. In this case, the digitizer returns a result code of `digiUnimpErr`.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDGetDataRate

The `VDGetDataRate` function allows an application to retrieve information that describes the performance capabilities of a video digitizer.

```
pascal VideoDigitizerError VDGetDataRate
                                   (VideoDigitizerComponent ci,
                                   long *milliSecPerFrame,
                                   Fixed *framesPerSecond,
                                   long *bytesPerSecond);
```

Video Digitizer Components

<code>ci</code>	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
<code>milliSecPerFrame</code>	Contains a pointer to a long integer. The video digitizer returns a value that indicates the number of milliseconds of synchronous overhead involved in digitizing a single frame. This value includes the average delay incurred between the time when the digitizer requests a frame from its associated device, and the time at which the device delivers the frame.
<code>framesPerSecond</code>	Contains a pointer to a fixed value. The video digitizer supplies the maximum rate at which it can capture video. Note that this value may differ from the rate that the application set with the <code>VDSetFrameRate</code> function, described in the previous section.
<code>bytesPerSecond</code>	Contains a pointer to a long integer. Video digitizers that can return compressed image data return a value that indicates the approximate number of bytes per second that the digitizer is generating compressed data, given the current compression settings and frame rate settings.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

Controlling Color

Video digitizer components support color digitization. Therefore, these components provide several functions that allow applications to control the color digitization process.

You can use `VDSetInputColorSpaceMode` in your application to enable and disable color digitization; you can use the `VDGetInputColorSpaceMode` function to determine whether color digitization is enabled. The `VDUseThisCLUT` function allows you to specify a color lookup table to be used by the video digitizer component. In cases where the component cannot accommodate a particular lookup table, your application can use the `VDGetCLUTInUse` function to retrieve the color lookup table used by the digitizer component.

Your application can determine whether a digitizer component supports color digitization by examining the input capability flags of the component. Specifically, if the `digiInDoesColor` flag is set to 1, the component supports color digitization. Applications can use the `VDGetCurrentFlags` function to obtain the input capability flags of a component (see “Getting Information About Video Digitizer Components” on page 8-24 for more information).

Your application can determine a digitizer's supported pixel depths by calling the `VDGetDMA Depths` function.

VDUseThisCLUT

Some video digitizer components allow applications to specify the lookup table for color digitization. Your application can set the color lookup table by calling the `VDUseThisCLUT` function.

```
pascal VideoDigitizerError VDUseThisCLUT
                                (VideoDigitizerComponent ci,
                                 CTabHandle colorTableHandle);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

colorTableHandle Contains a color table handle. The video digitizer component uses the color table referred to by this parameter.

DESCRIPTION

Applications can determine whether a digitizer component supports specified lookup tables by examining the digitizer component's output capability flags. Specifically, if the `digiOutDoesILUT` flag is set to 1, the digitizer component allows applications to specify color lookup tables. Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to obtain the input capability flags of a component.

This feature is only useful for capturing 8-bit color video.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDGetCLUTInUse

The `VDGetCLUTInUse` function allows an application to obtain the color lookup table used by a video digitizer component. By using the Palette Manager, the application can then set the destination so that it uses the same lookup table.

```
pascal VideoDigitizerError VDGetCLUTInUse
                                (VideoDigitizerComponent ci,
                                 CTabHandle *colorTableHandle);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

Video Digitizer Components

colorTableHandle

Contains a pointer to a field that is to receive a color table handle. The video digitizer component returns a handle to its color lookup table. Applications can then set the destination to use this returned color table. Your application is responsible for disposing of this handle.

DESCRIPTION

In general, applications use this function only when a video digitizer component does not allow applications to specify lookup tables with the `VDUseThisCLUT` function. Applications can determine whether a digitizer component supports specified lookup tables by examining the component's output capability flags. Specifically, if the `digiOutDoesILUT` flag is set to 1, the digitizer component allows applications to specify color lookup tables. Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to obtain the input capability flags of a component.

RESULT CODES

<code>noErr</code>	0	No error
<code>memFullErr</code>	-108	Not enough room in heap zone
<code>digiUnimpErr</code>	-2201	Function not supported

VDSetInputColorSpaceMode

The `VDSetInputColorSpaceMode` function allows applications to choose between color and grayscale digitized video.

```
pascal VideoDigitizerError VDSetInputColorSpaceMode
                                (VideoDigitizerComponent ci,
                                 short colorSpaceMode);
```

`ci` Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

`colorSpaceMode`

Controls color digitization. The following values are valid:

0	Grayscale digitization
1	Color digitization

DESCRIPTION

Applications can determine whether a digitizer component supports grayscale or color digitization by examining the digitizer component's input capability flags. Specifically, if the `digiInDoesColor` flag is set to 1, the digitizer component supports color digitization. Similarly, if the `digiInDoesBW` flag is set to 1, the digitizer component supports grayscale digitization. Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to obtain the input capability flags of a digitizer component.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

VDGetInputColorSpaceMode

The `VDGetInputColorSpaceMode` function allows applications to determine whether a digitizer is operating in color or grayscale mode.

```
pascal VideoDigitizerError VDGetInputColorSpaceMode
                                (VideoDigitizerComponent ci,
                                 short *colorSpaceMode);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>colorSpaceMode</code>	Contains a pointer to a value that indicates whether the digitizer is operating in color or grayscale mode. The following values are valid: 0 Grayscale digitization 1 Color digitization

DESCRIPTION

Applications can determine whether a digitizer component supports grayscale or color digitization by examining the digitizer component's input capability flags. Specifically, if the `digiInDoesColor` flag is set to 1, the digitizer component supports color digitization. Similarly, if the `digiInDoesBW` flag is set to 1, the digitizer component supports grayscale digitization. Applications can use the `VDGetCurrentFlags` function (described on page 8-25) to obtain the input capability flags of a digitizer component.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported
qtParamErr	-2202	Invalid parameter value

SEE ALSO

Applications can choose between color and grayscale digitization by calling the `VDSetInputColorSpaceMode` function, which is described in the previous section.

VDGetDMADepths

The `VDGetDMADepths` function allows an application to determine which pixel depths a digitizer supports. This function is supported only by digitizers that support DMA (that is, their `digiOutDoesDMA` output capability flag is set to 1).

```
pascal VideoDigitizerError VDGetDMADepths
                                (VideoDigitizerComponent ci,
                                 long *depthArray,
                                 long *preferredDepth);
```

ci Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's `OpenComponent` function.

depthArray Contains a pointer to a long integer. The video digitizer returns a value that indicates the depths it can support. Each depth is represented by a single bit in this field. More than one bit may be set to 1.

preferredDepth Contains a pointer to a long integer. Video digitizers that have a preferred depth value return that value in this field, using one of the possible values of the `depthArray` parameter. Digitizers that do not prefer any given value set this field to 0.

DESCRIPTION

The flags returned by this function augment the information that an application can obtain from the digitizer's output capability flags in the digitizer information structure (see "Capability Flags" beginning on page 8-14 for more information). If a digitizer does not support this function but does support DMA, an application may assume that the digitizer can handle offscreen buffers at all of the depths indicated in its output capabilities flags.

Before a program that uses a video digitizer creates an offscreen buffer, it should call the `VDGetDMA Depths` function to determine the pixel depths supported by the digitizer. If possible, the program should use the preferred depth, in order to obtain the best possible display performance.

Applications may use the following enumerators to set bits in the field referred to by the `depthArray` parameter.

```
enum {
    dmaDepth1      = 1, /* supports black and white */
    dmaDepth2      = 2, /* supports 2-bit color */
    dmaDepth4      = 4, /* supports 4-bit color */
    dmaDepth8      = 8, /* supports 8-bit color */
    dmaDepth16     = 16, /* supports 16-bit color */
    dmaDepth32     = 32, /* supports 32-bit color */
    dmaDepth2Gray  = 64, /* supports 2-bit grayscale */
    dmaDepth4Gray  = 128, /* supports 4-bit grayscale */
    dmaDepth8Gray  = 256 /* supports 8-bit grayscale */
};
```

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

Controlling Analog Video

Some video digitizer components may provide functions that allow applications to control the characteristics of the input analog video signal. This section describes these analog video functions.

The `VDGetVideoDefaults` function returns the suggested default values for the analog video parameters that can be affected by functions described in this section.

A number of functions affect gamma correction. The `VDSetInputGammaRecord` and `VDGetInputGammaRecord` functions work with gamma structures (see *Designing Cards and Drivers for the Macintosh Family*, third edition, for more information about gamma structures). You can use the `VDSetInputGammaValue` and `VDGetInputGammaValue` functions to allow your application to set particular gamma values.

The `VDSetBlackLevelValue`, `VDGetBlackLevelValue`, `VDSetWhiteLevelValue`, and `VDGetWhiteLevelValue` functions allow applications to work with black levels and white levels in the source video. **Black level** refers to the degree of blackness in an image. This is a common setting on a video digitizer. The highest setting produces an all-black image; on the other hand, the lowest setting yields little, if any, black even with black objects in the scene. Black level is a significant setting because it can be adjusted so that there is little or no noise in an image. **White level** refers to the degree of whiteness in an image. It is also a common video digitizer setting.

Video Digitizer Components

The `VDSetContrast`, `VDGetContrast`, `VDSetSharpness`, and `VDGetSharpness` functions allow applications to work with contrast and sharpness values in the source video. The `VDGetBrightness` and `VDSetBrightness` functions allow applications to work with the image brightness setting.

The `VDSetHue`, `VDGetHue`, `VDSetSaturation`, and `VDGetSaturation` functions allow applications to work with hue and saturation settings in the source video.

VDGetVideoDefaults

The `VDGetVideoDefaults` function returns the recommended values for many of the analog video parameters that may be set by applications.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDGetVideoDefaults
                                (VideoDigitizerComponent ci,
                                 unsigned short *blackLevel,
                                 unsigned short *whiteLevel,
                                 unsigned short *brightness,
                                 unsigned short *hue,
                                 unsigned short *saturation,
                                 unsigned short *contrast,
                                 unsigned short *sharpness);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

blackLevel Contains a pointer to an integer that is to receive the default black level value. The video digitizer component places the default black level value into the field referred to by this parameter. Refer to the discussion of the `VDSetBlackLevelValue` function in the next section for more information about black level values.

whiteLevel Contains a pointer to an integer that is to receive the default white level value. The video digitizer component places the default white level value into the field referred to by this parameter. Refer to the discussion of the `VDSetWhiteLevelValue` function on page 8-69 for more information about white level values.

brightness Contains a pointer to an integer that is to receive the default brightness value. The video digitizer component places the default brightness value into the field referred to by this parameter. Refer to the discussion of the `VDSetBrightness` function on page 8-73 for more information about brightness values.

Video Digitizer Components

hue	Contains a pointer to an integer that is to receive the default hue value. The video digitizer component places the default hue value into the field referred to by this parameter. Refer to the discussion of the <code>VDSetHue</code> function on page 8-70 for more information about hue values.
saturation	Contains a pointer to an integer that is to receive the default saturation value. The video digitizer component places the default saturation value into the field referred to by this parameter. Refer to the discussion of the <code>VDSetSaturation</code> function on page 8-72 for more information about saturation values.
contrast	Contains a pointer to an integer that is to receive the default contrast value. The video digitizer component places the default contrast value into the field referred to by this parameter. Refer to the discussion of the <code>VDSetContrast</code> function on page 8-75 for more information about contrast values.
sharpness	Contains a pointer to an integer that is to receive the default sharpness value. The video digitizer component places the default sharpness value into the field referred to by this parameter. Refer to the discussion of the <code>VDSetSharpness</code> function on page 8-76 for more information about sharpness values.

RESULT CODE

noErr 0 No error

VDSetBlackLevelValue

The `VDSetBlackLevelValue` function sets the current black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

```
pascal VideoDigitizerError VDSetBlackLevelValue
                                (VideoDigitizerComponent ci,
                                 unsigned short *blackLevel);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

blackLevel Contains a pointer to an integer that contains the new black level value. The video digitizer component attempts to set the black level value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported
qtParamErr	-2202	Invalid parameter value

SEE ALSO

Applications can get the current black level value by calling the `VDGetBlackLevelValue` function (described in the next section). Applications can obtain the recommended black level value by calling the `VDGetVideoDefaults` function (described in the previous section).

VDGetBlackLevelValue

The `VDGetBlackLevelValue` function returns the current black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

```
pascal VideoDigitizerError VDGetBlackLevelValue
                                (VideoDigitizerComponent ci,
                                 unsigned short *blackLevel);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

blackLevel Contains a pointer to an integer that is to receive the current black level value. The video digitizer component places the black level value into the field referred to by this parameter.

DESCRIPTION

Applications can set the black level value by calling the `VDSetBlackLevelValue` function (described in the previous section). Applications can obtain the recommended black level value by calling the `VDGetVideoDefaults` function (described on page 8-66).

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

VDSetWhiteLevelValue

The `VDSetWhiteLevelValue` function sets the white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

```
pascal VideoDigitizerError VDSetWhiteLevelValue
                                (VideoDigitizerComponent ci,
                                 unsigned short *whiteLevel);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

whiteLevel Contains a pointer to an integer that contains the new white level value. The video digitizer component attempts to set the white level value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Applications can get the current white level value by calling the `VDGetWhiteLevelValue` function (described in the next section). Applications can obtain the recommended white level value by calling the `VDGetVideoDefaults` function (described on page 8-66).

VDGetWhiteLevelValue

The `VDGetWhiteLevelValue` function returns the current white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

```
pascal VideoDigitizerError VDGetWhiteLevelValue
                                (VideoDigitizerComponent ci,
                                 unsigned short *whiteLevel);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>whiteLevel</code>	Contains a pointer to an integer that is to receive the current white level value. The video digitizer component places the white level value into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the white level value by calling the `VDSetWhiteLevelValue` function (described in the previous section). Your application can obtain the recommended white level value by calling the `VDGetVideoDefaults` function (described on page 8-66).

VDSetHue

The `VDSetHue` function sets the current **hue value**. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0°, green is +120°, and blue is -120°). Video digitizer components support hue values that range from 0 (-180° shift in hue) to 65,535 (+179° shift in hue), where 32,767 represents a 0° shift in hue.

```
pascal VideoDigitizerError VDSetHue (VideoDigitizerComponent ci,
                                     unsigned short *hue);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>hue</code>	Contains a pointer to an integer that contains the new hue value. The video digitizer component attempts to set the hue value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Your application can obtain the current hue value by calling the `VDGetHue` function (described in the next section). To retrieve the recommended hue value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDGetHue

The `VDGetHue` function returns the current hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0°, green is +120°, and blue is -120°). Video digitizer components support hue values that range from 0 (-180° shift in hue) to 65,535 (+179° shift in hue), where 32,767 represents a 0° shift in hue.

```
pascal VideoDigitizerError VDGetHue (VideoDigitizerComponent ci,
                                     unsigned short *hue);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>hue</code>	Contains a pointer to an integer that is to receive the current hue value. The video digitizer component places the hue value into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the hue value by calling the `VDSetHue` function (described in the previous section). To obtain the recommended hue value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDSetSaturation

The `VDSetSaturation` function sets the **saturation value**, which controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

```
pascal VideoDigitizerError VDSetSaturation
                                (VideoDigitizerComponent ci,
                                 unsigned short *saturation);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

saturation Contains a pointer to an integer that contains the new saturation value. The video digitizer component attempts to set the saturation value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Applications can get the current saturation value by calling the `VDGetSaturation` function (described in the next section). Applications can obtain the recommended saturation value by calling the `VDGetVideoDefaults` function (described on page 8-66).

VDGetSaturation

The `VDGetSaturation` function returns the current saturation value, which controls color intensity. For example, at high saturation levels red appears to be red, while at low saturation red appears pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

```
pascal VideoDigitizerError VDGetSaturation
                                (VideoDigitizerComponent ci,
                                 unsigned short *saturation);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>saturation</code>	Contains a pointer to an integer that is to receive the current saturation value. The video digitizer component places the saturation value into the field referred to by this parameter.

DESCRIPTION

The `VDGetSaturation` function returns the current saturation value.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the saturation value by calling the `VDSetSaturation` function (described in the previous section). To obtain the recommended saturation value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDSetBrightness

The `VDSetBrightness` function sets the current brightness value, which controls the overall brightness of the digitized video image. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

```
pascal VideoDigitizerError VDSetBrightness
                                (VideoDigitizerComponent ci,
                                 unsigned short *brightness);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>brightness</code>	Contains a pointer to an integer that contains the new brightness value. The video digitizer component attempts to set the brightness value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

SEE ALSO

Your application can get the current brightness value by calling the `VDGetBrightness` function (described in the next section). To obtain the recommended brightness value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDGetBrightness

The `VDGetBrightness` function returns the current brightness value, which reflects the overall brightness of the digitized video image. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

```
pascal VideoDigitizerError VDGetBrightness
                                (VideoDigitizerComponent ci,
                                 unsigned short *brightness);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

brightness Contains a pointer to an integer that is to receive the current brightness value. The video digitizer component places the brightness value into the field referred to by this parameter.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

SEE ALSO

Your application can set the brightness value by calling the `VDSetBrightness` function (described in the previous section). To obtain the recommended brightness value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDSetContrast

The `VDSetContrast` function sets the current contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (that is, increase the slope of the transform).

```
pascal VideoDigitizerError VDSetContrast
                                (VideoDigitizerComponent ci,
                                 unsigned short *contrast);
```

- | | |
|-----------------------|---|
| <code>ci</code> | Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function. |
| <code>contrast</code> | Contains a pointer to an integer that contains the new contrast value. The video digitizer component attempts to set the contrast value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests. |

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Your application can obtain the current contrast value by calling the `VDGetContrast` function (described in the next section). To retrieve the recommended contrast value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDGetContrast

The `VDGetContrast` function returns the current contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (that is, increase the slope of the transform).

```
pascal VideoDigitizerError VDGetContrast
                                (VideoDigitizerComponent ci,
                                 unsigned short *contrast);
```

- | | |
|-----------------|---|
| <code>ci</code> | Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function. |
|-----------------|---|

Video Digitizer Components

contrast Contains a pointer to an integer that is to receive the current contrast value. The video digitizer component places the contrast value into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the contrast value by calling the `VDSetContrast` function (described in the previous section). To obtain the recommended contrast value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDSetSharpness

The `VDSetSharpness` function sets the sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

```
pascal VideoDigitizerError VDSetSharpness
                                (VideoDigitizerComponent ci,
                                 unsigned short *sharpness);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

sharpness Contains a pointer to an integer that contains the new sharpness value. The video digitizer component attempts to set the sharpness value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

SEE ALSO

Your application can obtain the current sharpness value by calling the `VDGetSharpness` function (described in the next section). To retrieve the recommended sharpness value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDGetSharpness

The `VDGetSharpness` function returns the current sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

```
pascal VideoDigitizerError VDGetSharpness
                                (VideoDigitizerComponent ci,
                                unsigned short *sharpness);
```

`ci` Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

`sharpness` Contains a pointer to an integer that is to receive the current sharpness value. The video digitizer component places the sharpness value into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the sharpness value by calling the `VDSetSharpness` function (described in the previous section). To obtain the recommended sharpness value, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDSetInputGammaRecord

The `VDSetInputGammaRecord` function allows an application to change the active input gamma data structure. Gamma structures give applications complete control over color filtering transforms.

```
pascal VideoDigitizerError VDSetInputGammaRecord
    (VideoDigitizerComponent ci,
     VDGamRecPtr inputGammaPtr);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

inputGammaPtr Contains a pointer to an input gamma structure. The input gamma structure is defined by the `gammaTbl` data type. For more information about gamma structures, see *Designing Cards and Drivers for the Macintosh Family*, third edition. The video digitizer component uses the input gamma structure specified by this parameter.

SPECIAL CONSIDERATIONS

Note that the `VDSetInputGammaRecord` function may override the current gamma value and contrast settings if the video digitizer component uses a lookup table to implement brightness and contrast.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can get a pointer to the current input gamma structure by calling the `VDGetInputGammaRecord` function, which is described in the next section.

VDGetInputGammaRecord

The `VDGetInputGammaRecord` function allows your application to retrieve a pointer to the active input gamma structure. Gamma structures give applications complete control over color filtering transforms and are therefore more precise than the gamma values that can be set by calling the `VDSetInputGammaValue` function (described in the next section).

```
pascal VideoDigitizerError VDGetInputGammaRecord
    (VideoDigitizerComponent ci,
     VDGamRecPtr *inputGammaPtr);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

inputGammaPtr Contains a pointer to a field that is to receive a pointer to an input gamma structure. The input gamma structure is defined by the `gammaTbl` data type. For more information about gamma structures, see *Designing Cards and Drivers for the Macintosh Family*, third edition. The video digitizer component places a pointer to its input gamma structure into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the input gamma structure by calling the `VDSetInputGammaRecord` function, which is described in the previous section.

VDSetInputGammaValue

The `VDSetInputGammaValue` function sets the gamma values. These gamma values control the brightness of the input video signal. Your application can implement special color effects, such as turning off specific color channels, by calling this function.

```
pascal VideoDigitizerError VDSetInputGammaValue
                                (VideoDigitizerComponent ci,
                                 Fixed channel1,
                                 Fixed channel2,
                                 Fixed channel3);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>channel1</code>	Specifies the gamma value for the red component of the input video signal.
<code>channel2</code>	Specifies the gamma value for the green component of the input video signal.
<code>channel3</code>	Specifies the gamma value for the blue component of the input video signal.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can retrieve the current gamma values by calling the `VDGetInputGammaValue` function (described in the next section). To obtain the recommended gamma values, your application can call the `VDGetVideoDefaults` function (described on page 8-66).

VDGetInputGammaValue

The `VDGetInputGammaValue` function returns the current gamma values. These gamma values control the brightness of the input video signal.

```
pascal VideoDigitizerError VDGetInputGammaValue
                                (VideoDigitizerComponent ci,
                                 Fixed *channel1, Fixed *channel2,
                                 Fixed *channel3);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>channel1</code>	Contains a pointer to a fixed field that is to receive the gamma value for the red component of the input video signal. The video digitizer component places the appropriate gamma value into the field referred to by this parameter.
<code>channel2</code>	Contains a pointer to a fixed field that is to receive the gamma value for the green component of the input video signal. The video digitizer component places the appropriate gamma value into the field referred to by this parameter.
<code>channel3</code>	Contains a pointer to a fixed field that is to receive the gamma value for the blue component of the input video signal. The video digitizer component places the appropriate gamma value into the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Your application can set the gamma values by calling the `VDSetInputGammaValue` function (described in the previous section). To obtain the recommended gamma values, you can call the `VDGetVideoDefaults` function (described on page 8-66).

Selectively Displaying Video

Video digitizer components may support one of three methods of selectively displaying video on the screen of a Macintosh computer. The three methods are key colors, alpha channels, and blend masks. For a complete description of these techniques for selectively displaying video, see “About Video Digitizer Components,” which begins on page 8-3.

Your application can determine whether a video digitizer component supports selective video display by examining the component's digitizer information structure (described on page 8-20). Specifically, the `vdigType` field indicates the type of blending supported by the digitizer. Applications can use the `VDGetDigitizerInfo` function (described on page 8-24) to retrieve a component's digitizer information structure.

Some video digitizer components support the use of key colors as a mechanism for selectively displaying video on the screen of a Macintosh computer. When a key color is active, the digitizer component replaces all screen occurrences of that color with the appropriate portion of the source video. Video digitizer components that support key colors provide a number of functions to applications. Those functions are described in this section.

Video Digitizer Components

Your applications can use the `VDSetKeyColor`, `VDAddKeyColor`, and `VDSetKeyColorRange` functions to set one or more key colors for a video digitizer component. The `VDGetKeyColor`, `VDGetNextKeyColor`, and `VDGetKeyColorRange` functions allow your application to retrieve information about the currently active key colors.

Alpha channels and blend masks work similarly to one another. Digitizer components that support alpha channels use a portion of each pixel value to indicate the degree of video display for that pixel. Digitizer components that support blend masks use the mask to indicate the degree of video display for corresponding pixels.

Your applications can use the `VDGetMaskandValue` function to determine the appropriate mask value for a desired blend level. The `VDSetMasterBlendLevel` function allows applications to set a blend level that applies to the entire source video image. The `VDGetMaskPixMap` function allows applications to retrieve the pixel map that defines the blend mask.

VDSetKeyColor

The `VDSetKeyColor` function allows applications to set the key color.

All video digitizer components that support key colors must support this function.

```
pascal VideoDigitizerError VDSetKeyColor
                                (VideoDigitizerComponent ci,
                                 long index);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>index</code>	Specifies the new key color. The value of the <code>index</code> field corresponds to a color in the current color lookup table.

DESCRIPTION

Some video digitizer components support multiple key colors. The `VDSetKeyColor` function instructs such digitizer components to clear the key color list and insert a single entry for the specified color. Applications can then use the `VDAddKeyColor` function, described on page 8-84, to place additional colors into the key color list.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value

VDGetKeyColor

The `VDGetKeyColor` function allows your application to obtain the index value of the active key color.

All video digitizer components that support key colors must support this function.

```
pascal VideoDigitizerError VDGetKeyColor
                                (VideoDigitizerComponent ci,
                                 long *index);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
index	Contains a pointer to a field that is to receive the index of the key color. This index value identifies the key color within the currently active color lookup table. If there are several active key colors, the video digitizer returns the first color from the key color list. Subsequently, applications use the <code>VDGetNextKeyColor</code> function (described on page 8-86) to obtain other colors from the list. If there is no active key color, the <code>VDGetKeyColor</code> function sets the field to -1.

DESCRIPTION

In cases where there are several key colors, the `VDGetKeyColor` function always returns the index of the first color in the list. Applications should then use the `VDGetNextKeyColor` function (described on page 8-86) to retrieve the remaining colors in the list.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

VDSetKeyColorRange

Some video digitizer components that support key colors may allow applications to set a range of key color values. The key color range is expressed as a range of RGB color values. The `VDSetKeyColorRange` function allows your application to define a key color range.

```
pascal VideoDigitizerError VDSetKeyColorRange
                                (VideoDigitizerComponent ci,
                                 RGBColor *minRGB,
                                 RGBColor *maxRGB);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>minRGB</code>	Contains a pointer to a field that contains the lower bound of the key color range. All colors in the color table between the color specified by the <code>minRGB</code> parameter and the color specified by the <code>maxRGB</code> parameter are considered key colors.
<code>maxRGB</code>	Contains a pointer to a field that contains the upper bound of the key color range. All colors in the color table between the color specified by the <code>minRGB</code> parameter and the color specified by the <code>maxRGB</code> parameter are considered key colors.

DESCRIPTION

If the digitizer component cannot accommodate all the colors that are defined in the specified range, it returns a result value of `noMoreKeyColors`.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>noMoreKeyColors</code>	-2205	Key color list is full

SEE ALSO

Your application can obtain the current key color range by calling the `VDGetKeyColorRange` function, which is described on page 8-85.

VDAddKeyColor

Some video digitizer components can support more than one active key color. The `VDAddKeyColor` function allows applications to add a key color to a component's list of active key colors.

```
pascal VideoDigitizerError VDAddKeyColor
                                (VideoDigitizerComponent ci,
                                 long *index);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>index</code>	Contains a pointer to the color to add to the key color list. The value of the <code>index</code> field corresponds to a color in the current color lookup table.

DESCRIPTION

If the digitizer component cannot accommodate any more key colors, it returns a result code of `noMoreKeyColors`.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>qtParamErr</code>	-2202	Invalid parameter value
<code>noMoreKeyColors</code>	-2205	Key color list is full

SEE ALSO

To ensure that the key color list contains only the desired colors, your application should use the `VDSetKeyColor` function (described on page 8-82) to set the first key color.

VDGetKeyColorRange

Some video digitizer components that support key colors may allow applications to set a range of key color values. The key color range is expressed as a range of RGB color values. The `VDGetKeyColorRange` function allows applications to obtain the currently defined key color range.

```
pascal VideoDigitizerError VDGetKeyColorRange
                                (VideoDigitizerComponent ci,
                                 RGBColor *minRGB,
                                 RGBColor *maxRGB);
```

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>minRGB</code>	Contains a pointer to a field that is to receive the lower bound of the key color range. The video digitizer component places the RGB color that corresponds to the lower end of the range in the field referred to by this parameter.
<code>maxRGB</code>	Contains a pointer to a field that is to receive the upper bound of the key color range. The video digitizer component places the RGB color that corresponds to the upper end of the range in the field referred to by this parameter.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported
<code>badCallOrder</code>	-2209	Digitizer component not ready for this function

SEE ALSO

Your application can set the color range by calling the `VDSetKeyColorRange` function, which is described on page 8-83.

VDGetNextKeyColor

The `VDGetNextKeyColor` function allows your application to obtain the index value of the active key colors in cases where the digitizer component supports multiple key colors. Your application can use the `VDGetKeyColor` function (described on page 8-83) to retrieve the first key color in the list. Subsequently, your application can call the `VDGetNextKeyColor` function to retrieve the other colors in the key color list.

All video digitizer components that support multiple key colors must support this function.

```
pascal VideoDigitizerError VDGetNextKeyColor
                                (VideoDigitizerComponent ci,
                                 long index);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

index Specifies a field that is to receive the index of the next key color. This index value identifies the key color within the currently active color lookup table. If there are no more colors left in the list, the digitizer component sets the field referred to by the `index` parameter to -1.

DESCRIPTION

The `VDGetNextKeyColor` function returns an index value of -1 when there are no more colors in the list.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDSetMasterBlendLevel

The `VDSetMasterBlendLevel` function allows your application to set the blend level value for the input video signal. This value applies to the entire source video image.

```
pascal VideoDigitizerError VDSetMasterBlendLevel
                                   (VideoDigitizerComponent ci,
                                   unsigned short *blendLevel);
```

- | | |
|------------|--|
| ci | Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function. |
| blendLevel | Contains a pointer to a field that specifies the new master blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all video. The digitizer component returns the new value in this field, so your application can avoid using unsupported values in future requests. |

RESULT CODES

- | | | |
|--------------|-------|------------------------|
| noErr | 0 | No error |
| digiUnimpErr | -2201 | Function not supported |

VDGetMaskandValue

The `VDGetMaskandValue` function allows your application to obtain the appropriate alpha channel or blend mask value for a desired level of video blending. Your application specifies a desired level of video blend.

```
pascal VideoDigitizerError VDGetMaskandValue
                                   (VideoDigitizerComponent ci,
                                   unsigned short blendLevel,
                                   long *mask, long *value);
```

- | | |
|------------|---|
| ci | Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function. |
| blendLevel | Specifies the desired blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all video. |

Video Digitizer Components

<code>mask</code>	Contains a pointer to a field that is to receive a value indicating which bits are meaningful in the data returned for the <code>value</code> parameter. The video digitizer component sets to 1 the bits that correspond to meaningful bits in the data returned for the <code>value</code> parameter.
<code>value</code>	Contains a pointer to a field that is to receive data that can be used to obtain the desired blend level. The data returned for the <code>mask</code> parameter indicates which bits are valid in the data returned for this parameter.

DESCRIPTION

The video digitizer returns the corresponding mask value. The application can then use this value to set the alpha channel or blend mask.

The information returned by the digitizer component differs based on the type of blending supported by the component. In all cases, however, the returned value of the `value` parameter contains the value for the desired blend level, and the returned value of the `mask` parameter indicates which bits in the `value` parameter are meaningful. Bits in the returned `mask` parameter value that are set to 1 correspond to meaningful bits in the returned `value` parameter value.

For example, if an application requests a 50 percent video blend level from a digitizer that supports 8-bit alpha channels, the digitizer component might return the following values:

<code>mask</code>	<code>0xFF000000</code>	Identifies full upper byte as the alpha channel
<code>value</code>	<code>0x80000000</code>	Value for 50 percent blend level

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDGetMaskPixMap

The `VDGetMaskPixMap` function allows applications to retrieve the pixel map data for a component's blend mask. This function is supported only by digitizer components that support blend masks.

```
pascal VideoDigitizerError VDGetMaskPixMap
                                (VideoDigitizerComponent ci,
                                 PixMapHandle maskPixMap);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
<code>maskPixMap</code>	Contains a handle to a pixel map. The video digitizer component returns the pixel map data for its blend mask into the pixel map specified by this parameter. The video digitizer component resizes the handle as appropriate. Your application is responsible for disposing of this handle.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

Clipping

Some video digitizer components can clip the output video image based on an arbitrary clipping region. Your application can determine whether a video digitizer component supports clipping by examining the digitizer information structure of the component. Specifically, if the `digiOutDoesMask` flag is set to 1 in the `outputCapabilityFlags` field of the appropriate digitizer information structure, the component supports clipping. See “The Digitizer Information Structure” beginning on page 8-20 for details. Your application can obtain a component's digitizer information structure by calling the `VDGetDigitizerInfo` function, which is described on page 8-24. This section describes the functions provided to applications by components that support clipping.

Applications can use the `VDSetClipState` and `VDGetClipState` functions to enable and disable clipping, and to determine whether clipping is enabled. Applications can use the `VDSetClipRgn` and `VDClearClipRgn` functions to manipulate the clipping region. Applications can use these functions only during an active grab sequence. Applications set the initial clipping settings by calling either `VDSetPlayThruDestination` or `VDSetPlayThruGlobalRect` (described on page 8-35 and page 8-39, respectively).

Note

The functions that manipulate clipping and clipping state operate on a clipping region in addition to the one specified by the mask passed by the `VDSetPlayThruDestination` and `VDSetUpBuffers` functions (described on page 8-35 and page 8-54, respectively). To determine the final clipping regions, intersect these two clippings. ♦

VDSetsClipRgn

The VDSetsClipRgn function allows your application to define a clipping region.

```
pascal VideoDigitizerError VDSetsClipRgn
                                (VideoDigitizerComponent ci,
                                RgnHandle clipRegion);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's OpenComponent function.

clipRegion Specifies the clipping region.

DESCRIPTION

When clipping is enabled, the video digitizer component performs clipping in the region specified with this function.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

SEE ALSO

Applications can disable all or part of a clipping region by calling the VDClearClipRgn function, described in the next section.

VDClearClipRgn

The VDClearClipRgn function allows your application to disable all or part of a clipping region that was previously set with the VDSetsClipRgn function, which is described in the previous section.

```
pascal VideoDigitizerError VDClearClipRgn
                                (VideoDigitizerComponent ci,
                                RgnHandle clipRegion);
```


Video Digitizer Components

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
clipRegion	Specifies the clipping region to clear. This region must correspond to all or part of the clipping region established previously with the <code>VDSetClipRgn</code> function.

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

VDSetClipState

The `VDSetClipState` function allows applications to control whether clipping is enabled.

```
pascal VideoDigitizerError VDSetClipState
                                (VideoDigitizerComponent ci,
                                 short clipEnable);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
clipEnable	Controls whether clipping is enabled. Valid values are
0	Disable clipping
1	Enable clipping

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported

SEE ALSO

Applications can determine whether clipping is enabled by calling the `VDGetClipState` function, which is described in the next section.

VDGetClipState

The `VDGetClipState` function allows applications to determine whether clipping is enabled.

```
pascal VideoDigitizerError VDGetClipState
                                (VideoDigitizerComponent ci,
                                 short *clipEnable);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

clipEnable Contains a pointer to a field that is to receive a value indicating whether clipping is enabled. The video digitizer component places one of the following values into the field referred to by the `clipEnable` parameter:

0	Clipping disabled
1	Clipping enabled

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

SEE ALSO

Applications can enable and disable clipping by calling the `VDSetClipState` function, described in the previous section.

Utility Functions

This section describes a number of utility functions that may be supported by some video digitizer components.

The `VDSetPLLFilterType` and `VDGetPLLFilterType` functions allow applications to control which **phase-locked loop (PLL)** is used by a video digitizer component that supports multiple PLLs.

The `VDSetFieldPreference` and `VDGetFieldPreference` functions allow applications to control which field is used for some vertical scaling operations.

The `VDSetDigitizerUserInterrupt` function allows applications to install custom interrupt functions that are called by the video digitizer component.

The `VDGetSoundInputDriver` function allows an application to retrieve information about a digitizer's sound input driver.

The `VDGetPreferredTimeScale` function allows an application to determine a digitizer's preferred time scale.

VDSetPLLFilterType

The `VDSetPLLFilterType` function allows applications to specify which PLL is to be active.

```
pascal VideoDigitizerError VDSetPLLFilterType
                                (VideoDigitizerComponent ci,
                                 short pllType);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
pllType	Indicates which PLL is to be active. Available values are
0	Broadcast mode
1	VTR mode (stands for video tape recorder—equivalent to VCR, which stands for video cassette recorder)

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported
qtParamErr	-2202	Invalid parameter value

SEE ALSO

Applications can get the active PLL type by calling the `VDGetPLLFilterType` function, which is described in the next section.

VDGetPLLFilterType

The `VDGetPLLFilterType` function allows applications to determine which PLL is currently active.

```
pascal VideoDigitizerError VDGetPLLFilterType
                                (VideoDigitizerComponent ci,
                                 short *pllType);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
pllType	Points to a field that is to receive a value indicating which PLL is active. Available values are
0	Broadcast mode
1	VTR mode

RESULT CODES

noErr	0	No error
digiUnimpErr	-2201	Function not supported
qtParamErr	-2202	Invalid parameter value

SEE ALSO

Applications can set the PLL type by calling the `VDSetPLLFilterType` function, which is described in the previous section.

VDSetFieldPreference

The `VDSetFieldPreference` function allows applications to specify which field to use in cases where the vertical scaling is less than half size.

All video digitizer components must support this function.

```
pascal VideoDigitizerError VDSetFieldPreference
                                (VideoDigitizerComponent ci,
                                 short fieldFlag);
```

ci Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

fieldFlag Indicates which field to use. Valid values are

`vdUseAnyField`

Digitizer component decides which field to use

`vdUseOddField`

Digitizer uses odd field

`vdUseEvenField`

Digitizer uses even field

DESCRIPTION

Applications can specify that the digitizer use either the odd-line field or the even-line field; alternatively, applications can let the component decide which field to use.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDGetFieldPreference

The `VDGetFieldPreference` function allows applications to determine which field is being used in cases where the image is vertically scaled to half its original size.

```
pascal VideoDigitizerError VDGetFieldPreference
                                   (VideoDigitizerComponent ci,
                                   short *fieldFlag);
```

ci	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.
fieldFlag	Points to a field that is to receive a value indicating which field is being used. Valid values are <code>vdUseAnyField</code> Digitizer component decides which field to use <code>vdUseOddField</code> Digitizer component uses odd field <code>vdUseEvenField</code> Digitizer component uses even field

DESCRIPTION

Video digitizer components can use either the odd-line field or the even-line field. All video digitizer components must support this function.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

VDSetDigitizerUserInterrupt

The `VDSetDigitizerUserInterrupt` function allows applications to set custom interrupt functions.

```
pascal VideoDigitizerError VDSetDigitizerUserInterrupt
                                   (VideoDigitizerComponent ci,
                                   long flags,
                                   VdigIntProc userInterruptProc,
                                   long refcon);
```

Video Digitizer Components

<code>ci</code>	Specifies the video digitizer component for the request. Applications obtain this reference from the Component Manager's <code>OpenComponent</code> function.				
<code>flags</code>	Indicates when the interrupt function is to be called. Applications may set more than one flag to 1. The following flags are defined: <table> <tr> <td><code>Bit 0</code></td><td>Calls the interrupt function on even-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an even-line field.</td></tr> <tr> <td><code>Bit 1</code></td><td>Calls the interrupt function on odd-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an odd-line field.</td></tr> </table>	<code>Bit 0</code>	Calls the interrupt function on even-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an even-line field.	<code>Bit 1</code>	Calls the interrupt function on odd-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an odd-line field.
<code>Bit 0</code>	Calls the interrupt function on even-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an even-line field.				
<code>Bit 1</code>	Calls the interrupt function on odd-line fields. If this flag is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an odd-line field.				
<code>userInterruptProc</code>	Contains a pointer to the custom interrupt function. Applications set this parameter to <code>nil</code> to remove a custom interrupt function. Every custom interrupt function must support the following interface: <pre>pascal void MyInterruptProc (long flags, long refcon);</pre> See page 8-98 for details on the parameters of the <code>MyInterruptProc</code> function.				
<code>refcon</code>	Contains parameter data that is appropriate for the interrupt procedure.				

DESCRIPTION

The video digitizer component calls these custom interrupt functions during field or frame interrupt processing. The application function can then perform special processing.

RESULT CODES

<code>noErr</code>	0	No error
<code>digiUnimpErr</code>	-2201	Function not supported

VDGetSoundInputDriver

The `VDGetSoundInputDriver` function allows an application to retrieve information about a digitizer's sound input driver.

```
pascal VideoDigitizerError VDGetSoundInputDriver
                                (VideoDigitizerComponent ci,
                                 Str255 soundDriverName);
```

Video Digitizer Components

<code>ci</code>	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
<code>soundDriverName</code>	Specifies a pointer to a string. The video digitizer returns the name of its sound input driver. If the digitizer does not have an associated driver, it returns a result code of <code>digiUnimpErr</code> .

DESCRIPTION

An application can use the driver name returned by this function to choose an appropriate sound input device to use with this digitizer.

RESULT CODES

<code>noErr</code>	0	No error
<code>qtParamErr</code>	-2202	Invalid parameter value

VDGetPreferredTimeScale

The `VDGetPreferredTimeScale` function allows an application to determine a digitizer's preferred time scale.

```
pascal VideoDigitizerError VDGetPreferredTimeScale
                                (VideoDigitizerComponent ci,
                                TimeScale *preferred);
```

<code>ci</code>	Identifies the application's connection to the video digitizer component. An application obtains this value from the Component Manager's <code>OpenComponent</code> function.
<code>preferred</code>	Contains a pointer to a time scale structure. The video digitizer returns information about its preferred time scale.

DESCRIPTION

Apple's sequence grabber component uses this function to establish the time scale of the media that it creates from the digitizer's output. This is especially beneficial for digitizers that return compressed data, because it allows these digitizers to timestamp the frames very accurately.

If the digitizer does not have a preferred time scale, it returns a result code of `digiUnimpErr`.

RESULT CODES

noErr	0	No error
qtParamErr	-2202	Invalid parameter value

Application-Defined Function

Applications can provide a custom interrupt function in the `userInterruptProc` parameter of the `VDSetDigitizerUserInterrupt` function. Every custom interrupt function must support the following interface:

```
pascal void MyInterruptProc (long flags, long refcon);
```

flags	Indicates when the interrupt function has been called. The video digitizer component sets these flags to indicate the circumstances in which the function has been called. The following flags are defined:	
	Bit 0	Even-line field interrupt. If this flag is set to 1, the video digitizer component is about to display an even-line field.
	Bit 1	Odd-line field interrupt. If this flag is set to 1, the video digitizer component is about to display an odd-line field.
refcon	Contains parameter data that is appropriate for the interrupt function. The application assigns the value of the reference constant when it sets the interrupt function.	

Summary of Video Digitizer Components

C Summary

Constants

```
enum {
    videoDigitizerComponentType = 'vdig', /* standard type for video
                                           digitizer components */

    /* input format standards */
    ntscIn          = 0, /* National Television System Committee */
    palIn           = 1, /* Phase Alternation Line */
    secamIn         = 2, /* Sequential Color with Memory */

    /* input formats */
    compositeIn     = 0, /* no color separation of channels */
    sVideoIn       = 1, /* s-video (super VHS) */
    rgbComponentIn  = 2, /* separate channels for red, green, & blue */

    /* video digitizer component PlayThru states */
    vdPlayThruOff   = 0, /* playthrough off */
    vdPlayThruOn    = 1, /* playthrough on */

    /* field preference options in VDGetFieldPreference function */
    vdUseAnyField   = 0, /* digitizer component decides which field to use */
    vdUseOddField   = 1, /* digitizer component uses odd field */
    vdUseEvenField  = 2, /* digitizer component uses even field */

    /* input color space modes */
    vdDigitizerBW   = 0, /* digitizer component uses black and white */
    vdDigitizerRGB  = 1, /* digitizer component uses red, green, & blue */

    /* phase lock loop modes */
    vdBroadcastMode = 0, /* broadcast (laser disk) video mode */
    vdVTRMode       = 1, /* VCR (magnetic media) mode */
}
```

Video Digitizer Components

```

/* video digitizer component types */
vdTypeBasic = 0, /* basic component does not support clipping */
vdTypeAlpha = 1, /* component supports clipping with alpha channel */
vdTypeMask  = 2, /* component supports clipping with mask plane */
vdTypeKey   = 3, /* component supports clipping with one or more key
                  colors */

/* digitizer input capability/current flags */
digiInDoesNTSC      = (1L<<0), /* NTSC input */
digiInDoesPAL       = (1L<<1), /* PAL input */
digiInDoesSECAM     = (1L<<2), /* SECAM format */
digiInDoesGenLock   = (1L<<7), /* digitizer performs genlock */
digiInDoesComposite = (1L<<8), /* composite input */
digiInDoesSVideo    = (1L<<9), /* s-video input type */
digiInDoesComponent = (1L<<10), /* component (RGB) input type */
digiInVTR_Broadcast = (1L<<11), /* differentiates between magnetic
                                   media and broadcast input */
digiInDoesColor     = (1L<<12), /* digitizer supports color */
digiInDoesBW        = (1L<<13), /* digitizer supports black & white */

/* digitizer input current flags (these are valid only during active
   operating conditions) */
digiInSignalLock    = (1L<<31), /* digitizer detects locked input signal
                                   - this bit =
                                   horiz lock || vertical lock */

/* digitizer output capability/current flags */
digiOutDoes1        = (1L<<0), /* digitizer supports 1-bit pixels */
digiOutDoes2        = (1L<<1), /* digitizer supports 2-bit pixels */
digiOutDoes4        = (1L<<2), /* digitizer supports 4-bit pixels */
digiOutDoes8        = (1L<<3), /* digitizer supports 8-bit pixels */
digiOutDoes16       = (1L<<4), /* digitizer supports 16-bit pixels */
digiOutDoes32       = (1L<<5), /* digitizer supports 32-bit pixels */
digiOutDoesDither   = (1L<<6), /* digitizer dithers in indexed modes */
digiOutDoesStretch  = (1L<<7), /* digitizer can arbitrarily stretch */
digiOutDoesShrink   = (1L<<8), /* digitizer can arbitrarily shrink */

digiOutDoesMask     = (1L<<9), /* masks to clipping regions */
digiOutDoesDouble   = (1L<<11), /* stretches to exactly double size */
digiOutDoesQuad     = (1L<<12), /* stretches to exactly quadruple size */
digiOutDoesQuarter  = (1L<<13), /* shrinks to exactly one-quarter size */
digiOutDoesSixteenth = (1L<<14), /* shrinks to exactly one-sixteenth */
digiOutDoesRotate   = (1L<<15), /* supports rotation transformations */
digiOutDoesHorizFlip = (1L<<16), /* supports horizontal flips Sx < 0 */

```

Video Digitizer Components

```

digiOutDoesVertFlip = (1L<<17),/* supports vertical flips Sy < 0 */
digiOutDoesSkew     = (1L<<18),/* supports skew (shear,twist) */
digiOutDoesBlend    = (1L<<19),/* supports blend operations */
digiOutDoesWarp     = (1L<<20),/* supports warp operations */
digiOutDoesHW_DMA   = (1L<<21),/* not constrained to local device */
digiOutDoesHWPlayThru= (1L<<22),/* doesn't need time to play */
digiOutDoesILUT     = (1L<<23),/* does lookup table for index modes */
digiOutDoesKeyColor = (1L<<24),/* performs key color functions too */
digiOutDoesAsyncGrabs= (1L<<25),/* supports asynchronous grabs */
digiOutDoesUnreadableScreenBits
                    = (1L<<26),/* playthru doesn't generate readable
                               bits on screen */
digiOutDoesCompress = (1L<<27),/* supports compressed source devices */
digiOutDoesCompressOnly
                    = (1L<<28),/* can't draw images */
digiOutDoesPlayThruDuringCompress
                    = (1L<<29) /* can play while providing compressed
                               data */
};

enum {
    /* video digitizer interface */
    kSelectVDGetMaxSrcRect          = 0x1,/* VDGetMaxSrcRect (required) */
    kSelectVDGetActiveSrcRect      = 0x2,/* VDGetActiveSrcRect
                                         (required) */
    kSelectVDSaveDigitizerRect    = 0x3,/* VDSaveDigitizerRect
                                         (required) */
    kSelectVDGetDigitizerRect      = 0x4,/* VDGetDigitizerRect
                                         (required) */
    kSelectVDGetVBlankRect         = 0x5,/* VDGetVBlankRect (required) */
    kSelectVDGetMaskPixMap         = 0x6,/* VDGetMaskPixMap */

    /* 1 available selector here */
    kSelectVDGetPlayThruDestination = 0x8,/* VDGetPlayThruDestination
                                         (required) */
    kSelectVDUseThisCLUT           = 0x9,/* VDUseThisCLUT */
    kSelectVDSaveInputGammaValue   = 0xA,/* VDSaveInputGammaValue */
    kSelectVDGetInputGammaValue    = 0xB,/* VDGetInputGammaValue */
    kSelectVDSaveBrightness        = 0xC,/* VDSaveBrightness */
    kSelectVDGetBrightness         = 0xD,/* VDGetBrightness */
    kSelectVDSaveContrast          = 0xE,/* VDSaveContrast */
    kSelectVDSaveHue               = 0xF,/* VDSaveHue */
    kSelectVDSaveSharpness         = 0x10,/* VDSaveSharpness */
    kSelectVDSaveSaturation        = 0x11,/* VDSaveSaturation */

```

Video Digitizer Components

```

kSelectVDGetContrast          = 0x12, /* VDGetContrast */
kSelectVDGetHue              = 0x13, /* VDGetHue */
kSelectVDGetSharpness       = 0x14, /* VDGetSharpness */
kSelectVDGetSaturation       = 0x15, /* VDGetSaturation */
kSelectVDGrabOneFrame        = 0x16, /* VGrabOneFrame
                                     (required) */
kSelectVDGetMaxAuxBuffer     = 0x17, /* VDGetMaxAuxBuffer */
kSelectVDGetDigitizerInfo    = 0x19, /* VDGetDigitizerInfo
                                     (required) */
kSelectVDGetCurrentFlags     = 0x1A, /* VDGetCurrentFlags
                                     (required) */
kSelectVDSetKeyColor         = 0x1B, /* VDSetKeyColor */
kSelectVDGetKeyColor         = 0x1C, /* VDGetKeyColor */
kSelectVDAddKeyColor         = 0x1D, /* VAddKeyColor */
kSelectVDGetNextKeyColor     = 0x1E, /* VDGetNextKeyColor */
kSelectVDSetKeyColorRange    = 0x1F, /* VDSetKeyColorRange */
kSelectVDGetKeyColorRange    = 0x20, /* VDGetKeyColorRange */
kSelectVDSetDigitizerUserInterrupt = 0x21,
                               /* VDSetDigitizerUserInterrupt */
kSelectVDSetInputColorSpaceMode = 0x22, /* VDSetInputColorSpaceMode */
kSelectVDGetInputColorSpaceMode = 0x23, /* VDGetInputColorSpaceMode */
kSelectVDSetClipState       = 0x24, /* VDSetClipState */
kSelectVDGetClipState       = 0x25, /* VDGetClipState */
kSelectVDSetClipRgn         = 0x26, /* VDSetClipRgn */
kSelectVDClearClipRgn       = 0x27, /* VDClearClipRgn */
kSelectVDGetCLUTInUse       = 0x28, /* VDGetCLUTInUse */
kSelectVDSetPLLFilterType   = 0x29, /* VDSetPLLFilterType */
kSelectVDGetPLLFilterType   = 0x2A, /* VDGetPLLFilterType */
kSelectVDGetMaskandValue    = 0x2B, /* VDGetMaskandValue */
kSelectVDSetMasterBlendLevel = 0x2C, /* VDSetMasterBlendLevel */
kSelectVDSetPlayThruDestination = 0x2D, /* VDSetPlayThruDestination */
kSelectVDSetPlayThruOnOff   = 0x2E, /* VDSetPlayThruOnOff */
kSelectVDSetFieldPreference = 0x2F, /* VDSetFieldPreference
                                     (required) */
kSelectVDGetFieldPreference = 0x30, /* VDGetFieldPreference
                                     (required) */
kSelectVDPreflightDestination = 0x32, /* VDPreflightDestination
                                     (required) */
kSelectVDPreflightGlobalRect = 0x33, /* VDPreflightGlobalRect */
kSelectVDSetPlayThruGlobalRect = 0x34, /* VDSetPlayThruGlobalRect */
kSelectVDSetInputGammaRecord = 0x35, /* VDSetInputGammaRecord */
kSelectVDGetInputGammaRecord = 0x36, /* VDGetInputGammaRecord */
kSelectVDSetBlackLevelValue  = 0x37, /* VDSetBlackLevelValue */

```

Video Digitizer Components

```

kSelectVDGetBlackLevelValue      = 0x38, /* VDGetBlackLevelValue */
kSelectVDSetWhiteLevelValue      = 0x39, /* VDSetWhiteLevelValue */
kSelectVDGetWhiteLevelValue      = 0x3A, /* VDGetWhiteLevelValue */
kSelectVDGetVideoDefaults        = 0x3B, /* VDGetVideoDefaults */
kSelectVDGetNumberOfInputs       = 0x3C, /* VDGetNumberOfInputs */
kSelectVDGetInputFormat          = 0x3D, /* VDGetInputFormat */
kSelectVDSetInput                 = 0x3E, /* VDSetInput */
kSelectVDGetInput                 = 0x3F, /* VDGetInput */
kSelectVDSetInputStandard         = 0x40, /* VDSetInputStandard */
kSelectVDSetupBuffers            = 0x41, /* VDSetupBuffers */
kSelectVDGrabOneFrameAsync       = 0x42, /* VDGrabOneFrameAsync */
kSelectVDDone                    = 0x43, /* VDDone */
kSelectVDSetCompression          = 0x44, /* VDSetCompression */
kSelectVDCompressOneFrameAsync   = 0x45, /* VDCompressOneFrameAsync */
kSelectVDCompressDone            = 0x46, /* VDCompressDone */
kSelectVDReleaseCompressBuffer   = 0x47, /* VDReleaseCompressBuffer */
kSelectVDGetImageDescription     = 0x48, /* VDGetImageDescription */
kSelectVDResetCompressSequence   = 0x49, /* VDResetCompressSequence */
kSelectVDSetCompressionOnOff     = 0x4A, /* VDSetCompressionOnOff */
kSelectVDGetCompressionTypes     = 0x4B, /* VDGetCompressionTypes */
kSelectVDSetTimeBase             = 0x4C, /* VDSetTimeBase */
kSelectVDSetFrameRate            = 0x4D, /* VDSetFrameRate */
kSelectVDGetDataRate             = 0x4E, /* VDGetDataRate */
kSelectVDGetSoundInputDriver     = 0x4F, /* VDGetSoundInputDriver */
kSelectVDGetDMADepths            = 0x50, /* VDGetDMADepths */
kSelectVDGetPreferredTimeScale   = 0x51, /* VDGetPreferredTimeScale */
kSelectVDReleaseAsyncBuffers     = 0x52, /* VDReleaseAsyncBuffers */
};

/* flags for VDGetDMADepths depthArray parameter */
enum {
    dmaDepth1      = 1,      /* supports black and white */
    dmaDepth2      = 2,      /* supports 2-bit color */
    dmaDepth4      = 4,      /* supports 4-bit color */
    dmaDepth8      = 8,      /* supports 8-bit color */
    dmaDepth16     = 16,     /* supports 16-bit color */
    dmaDepth32     = 32,     /* supports 32-bit color */
    dmaDepth2Gray  = 64,     /* supports 2-bit grayscale */
    dmaDepth4Gray  = 128,    /* supports 4-bit grayscale */
    dmaDepth8Gray  = 256,    /* supports 8-bit grayscale */
};

```

Data Types

```

typedef ComponentInstance VideoDigitizerComponent; /* video digitizer
                                                    component */

typedef ComponentResult VideoDigitizerError;      /* video digitizer error */

struct DigitizerInfo {
    short    vdigType;          /* type of digitizer component */
    long     inputCapabilityFlags; /* input video signal features */
    long     outputCapabilityFlags; /* output digitized video data features
                                     of digitizer component */
    long     inputCurrentFlags;  /* status of input video signal */
    long     outputCurrentFlags; /* status of output digitized video data */
    short    slot;              /* temporary for connection purposes */
    GDHandle gdh;               /* temporary for digitizers with
                                     preferred screen */
    GDHandle maskgdh;           /* temporary for digitizers with
                                     mask planes */

    short    minDestHeight;     /* smallest resizable height */
    short    minDestWidth;      /* smallest resizable width */
    short    maxDestHeight;     /* largest resizable height */
    short    maxDestWidth;      /* largest resizable width */
    short    blendLevels;       /* number of blend levels supported
                                     (2 if 1 bit mask) */
    long     Private;           /* reserved--set to 0 */
};

typedef struct DigitizerInfo DigitizerInfo;

struct VdigBufferRecList {
    short    count;             /* # of buffers defined by this structure */
    MatrixRecordPtr matrix;     /* tranformation matrix applied to dest rects
                                     before video image is displayed */
    RgnHandle mask;             /* clip region applied to dest rect before
                                     video image is displayed */
    VdigBufferRec list[1]; /* array of output buffer specifications */
};

typedef struct {
    PixMapHandle dest;          /* handle to pixel map for destination buffer */
    Point         location;     /* location of video destination in pixel map */
    long          reserved;     /* reserved--set to 0 */
} VdigBufferRec;

```

Video Digitizer Components

```
typedef struct VDCompressionList {
    CodecComponent    codec;           /* component ID */
    CodecType         cType;           /* compressor type */
    Str63             typeName;        /* compression algorithm */
    Str63             name;            /* compressor name string */
    long              formatFlags;     /* data format flags */
    long              compressFlags;   /* capabilities flags */
    long              reserved;        /* set to 0 */
} VDCompressionList, *VDCompressionListPtr, **VDCompressionListHandle;
```

Video Digitizer Component Functions

Getting Information About Video Digitizer Components

```
pascal VideoDigitizerError VDGetDigitizerInfo
    (VideoDigitizerComponent ci,
     DigitizerInfo *info);

pascal VideoDigitizerError VDGetCurrentFlags
    (VideoDigitizerComponent ci,
     long *inputCurrentFlag,
     long *outputCurrentFlag);
```

Setting Source Characteristics

```
pascal VideoDigitizerError VDGetMaxSrcRect
    (VideoDigitizerComponent ci, short inputStd,
     Rect *maxSrcRect);

pascal VideoDigitizerError VDGetActiveSrcRect
    (VideoDigitizerComponent ci,
     short inputStd, Rect *activeSrcRect);

pascal VideoDigitizerError VDGetVBlankRect
    (VideoDigitizerComponent ci,
     short inputStd, Rect *vBlankRect);

pascal VideoDigitizerError VDSaveDigitizerRect
    (VideoDigitizerComponent ci,
     Rect *digitizerRect);

pascal VideoDigitizerError VDGetDigitizerRect
    (VideoDigitizerComponent ci,
     Rect *digitizerRect);
```

Selecting an Input Source

```

pascal VideoDigitizerError VGetNumberOfInputs
    (VideoDigitizerComponent ci, short *inputs);

pascal VideoDigitizerError VSetInput
    (VideoDigitizerComponent ci, short input);

pascal VideoDigitizerError VGetInput
    (VideoDigitizerComponent ci, short *input);

pascal VideoDigitizerError VGetInputFormat
    (VideoDigitizerComponent ci, short input,
     short *format);

pascal VideoDigitizerError VSetInputStandard
    (VideoDigitizerComponent ci,
     short inputStandard);

```

Setting Video Destinations

```

pascal VideoDigitizerError VSetPlayThruDestination
    (VideoDigitizerComponent ci,
     PixMapHandle dest, Rect *destRect,
     MatrixRecord *m, RgnHandle mask);

pascal VideoDigitizerError VDPreflightDestination
    (VideoDigitizerComponent ci,
     Rect *digitizerRect, PixMapHandle dest,
     Rect *destRect, MatrixRecord *m);

pascal VideoDigitizerError VGetPlayThruDestination
    (VideoDigitizerComponent ci,
     PixMapHandle *dest, Rect *destRect,
     MatrixRecord *m, RgnHandle *mask);

pascal VideoDigitizerError VSetPlayThruGlobalRect
    (VideoDigitizerComponent ci,
     GrafPtr theWindow, Rect *globalRect);

pascal VideoDigitizerError VDPreflightGlobalRect
    (VideoDigitizerComponent ci,
     GrafPtr theWindow, Rect *globalRect);

pascal VideoDigitizerError VGetMaxAuxBuffer
    (VideoDigitizerComponent ci,
     PixMapHandle *pm, Rect *r);

```

Controlling Compressed Source Devices

```

pascal VideoDigitizerError VGetCompressionTypes
    (VideoDigitizerComponent ci,
     VDCompressionListHandle h);

```


Video Digitizer Components

```

pascal VideoDigitizerError VDSetsCompression
    (VideoDigitizerComponent ci,
     OStype compressType, short depth,
     Rect *bounds, CodecQ spatialQuality,
     CodecQ temporalQuality, long keyFrameRate);
pascal VideoDigitizerError VDSetsCompressionOnOff
    (VideoDigitizerComponent ci, Boolean state);
pascal VideoDigitizerError VDCompressOneFrameAsync
    (VideoDigitizerComponent ci);
pascal VideoDigitizerError VDCompressDone
    (VideoDigitizerComponent ci, Boolean *done,
     Ptr *theData, long *dataSize,
     unsigned char *similarity, TimeRecord *t);
pascal VideoDigitizerError VDReleaseCompressBuffer
    (VideoDigitizerComponent ci, Ptr bufferAddr);
pascal VideoDigitizerError VDGetImageDescription
    (VideoDigitizerComponent ci,
     ImageDescriptionHandle desc);
pascal VideoDigitizerError VDRestartCompressSequence
    (VideoDigitizerComponent ci);
pascal VideoDigitizerError VDSetTimeBase
    (VideoDigitizerComponent ci, TimeBase t);

```

Controlling Digitization

```

pascal VideoDigitizerError VDSetsPlayThruOnOff
    (VideoDigitizerComponent ci, short state);
pascal VideoDigitizerError VDGrabOneFrame
    (VideoDigitizerComponent ci);
pascal VideoDigitizerError VDSetupBuffers
    (VideoDigitizerComponent ci,
     VdigBufferRecListHandle bufferList);
pascal VideoDigitizerError VDReleaseAsyncBuffers
    (VideoDigitizerComponent ci);
pascal VideoDigitizerError VDGrabOneFrameAsync
    (VideoDigitizerComponent ci, short buffer);
pascal long VDDone
    (VideoDigitizerComponent ci, short buffer);
pascal VideoDigitizerError VDSetFrameRate
    (VideoDigitizerComponent ci,
     Fixed framesPerSecond);
pascal VideoDigitizerError VDGetDataRate
    (VideoDigitizerComponent ci,
     long *milliSecPerFrame,
     Fixed *framesPerSecond, long *bytesPerSecond);

```

Controlling Color

```

pascal VideoDigitizerError VDUseThisCLUT
    (VideoDigitizerComponent ci,
     CTabHandle colorTableHandle);

pascal VideoDigitizerError VGetCLUTInUse
    (VideoDigitizerComponent ci,
     CTabHandle *colorTableHandle);

pascal VideoDigitizerError VSetInputColorSpaceMode
    (VideoDigitizerComponent ci,
     short colorSpaceMode);

pascal VideoDigitizerError VGetInputColorSpaceMode
    (VideoDigitizerComponent ci,
     short *colorSpaceMode);

pascal VideoDigitizerError VGetDMADepths
    (VideoDigitizerComponent ci,
     long *depthArray, long *preferredDepth);

```

Controlling Analog Video

```

pascal VideoDigitizerError VGetVideoDefaults
    (VideoDigitizerComponent ci,
     unsigned short *blackLevel,
     unsigned short *whiteLevel,
     unsigned short *brightness,
     unsigned short *hue,
     unsigned short *saturation,
     unsigned short *contrast,
     unsigned short *sharpness);

pascal VideoDigitizerError VSetBlackLevelValue
    (VideoDigitizerComponent ci,
     unsigned short *blackLevel);

pascal VideoDigitizerError VGetBlackLevelValue
    (VideoDigitizerComponent ci,
     unsigned short *blackLevel);

pascal VideoDigitizerError VSetWhiteLevelValue
    (VideoDigitizerComponent ci,
     unsigned short *whiteLevel);

pascal VideoDigitizerError VGetWhiteLevelValue
    (VideoDigitizerComponent ci,
     unsigned short *whiteLevel);

pascal VideoDigitizerError VSetHue
    (VideoDigitizerComponent ci,
     unsigned short *hue);

```

Video Digitizer Components

```

pascal VideoDigitizerError VGetHue
    (VideoDigitizerComponent ci,
     unsigned short *hue);
pascal VideoDigitizerError VSetSaturation
    (VideoDigitizerComponent ci,
     unsigned short *saturation);
pascal VideoDigitizerError VGetSaturation
    (VideoDigitizerComponent ci,
     unsigned short *saturation);
pascal VideoDigitizerError VSetBrightness
    (VideoDigitizerComponent ci,
     unsigned short *brightness);
pascal VideoDigitizerError VGetBrightness
    (VideoDigitizerComponent ci,
     unsigned short *brightness);
pascal VideoDigitizerError VSetContrast
    (VideoDigitizerComponent ci,
     unsigned short *contrast);
pascal VideoDigitizerError VGetContrast
    (VideoDigitizerComponent ci,
     unsigned short *contrast);
pascal VideoDigitizerError VSetSharpness
    (VideoDigitizerComponent ci,
     unsigned short *sharpness);
pascal VideoDigitizerError VGetSharpness
    (VideoDigitizerComponent ci,
     unsigned short *sharpness);
pascal VideoDigitizerError VSetInputGammaRecord
    (VideoDigitizerComponent ci,
     VDGamRecPtr inputGammaPtr);
pascal VideoDigitizerError VGetInputGammaRecord
    (VideoDigitizerComponent ci,
     VDGamRecPtr *inputGammaPtr);
pascal VideoDigitizerError VSetInputGammaValue
    (VideoDigitizerComponent ci,
     Fixed channel1, Fixed channel2,
     Fixed channel3);
pascal VideoDigitizerError VGetInputGammaValue
    (VideoDigitizerComponent ci,
     Fixed *channel1, Fixed *channel2,
     Fixed *channel3);

```

Selectively Displaying Video

```

pascal VideoDigitizerError VDSaveKeyColor
    (VideoDigitizerComponent ci, long index);
pascal VideoDigitizerError VDGetKeyColor
    (VideoDigitizerComponent ci, long *index);
pascal VideoDigitizerError VDSaveKeyColorRange
    (VideoDigitizerComponent ci,
     RGBColor *minRGB, RGBColor *maxRGB);
pascal VideoDigitizerError VDAAddKeyColor
    (VideoDigitizerComponent ci, long *index);
pascal VideoDigitizerError VDGetKeyColorRange
    (VideoDigitizerComponent ci,
     RGBColor *minRGB, RGBColor *maxRGB);
pascal VideoDigitizerError VDGetNextKeyColor
    (VideoDigitizerComponent ci, long index);
pascal VideoDigitizerError VDSaveMasterBlendLevel
    (VideoDigitizerComponent ci,
     unsigned short *blendLevel);
pascal VideoDigitizerError VDGetMaskandValue
    (VideoDigitizerComponent ci,
     unsigned short blendLevel, long *mask,
     long *value);
pascal VideoDigitizerError VDGetMaskPixMap
    (VideoDigitizerComponent ci,
     PixMapHandle maskPixMap);

```

Clipping

```

pascal VideoDigitizerError VDSaveClipRgn
    (VideoDigitizerComponent ci,
     RgnHandle clipRegion);
pascal VideoDigitizerError VDClearClipRgn
    (VideoDigitizerComponent ci,
     RgnHandle clipRegion);
pascal VideoDigitizerError VDSaveClipState
    (VideoDigitizerComponent ci, short clipEnable);
pascal VideoDigitizerError VDGetClipState
    (VideoDigitizerComponent ci, short *clipEnable);

```

Utility Functions

```

pascal VideoDigitizerError VDSavePLLFilterType
    (VideoDigitizerComponent ci, short pllType);
pascal VideoDigitizerError VDGetPLLFilterType
    (VideoDigitizerComponent ci, short *pllType);
pascal VideoDigitizerError VDSaveFieldPreference
    (VideoDigitizerComponent ci, short fieldFlag);
pascal VideoDigitizerError VDGetFieldPreference
    (VideoDigitizerComponent ci, short *fieldFlag);
pascal VideoDigitizerError VDSaveDigitizerUserInterrupt
    (VideoDigitizerComponent ci, long flags,
     VdigIntProc userInterruptProc, long refcon);
pascal VideoDigitizerError VDGetSoundInputDriver
    (VideoDigitizerComponent ci,
     Str255 soundDriverName);
pascal VideoDigitizerError VDGetPreferredTimeScale
    (VideoDigitizerComponent ci,
     TimeScale *preferred);

```

Application-Defined Function

```

pascal void MyInterruptProc (long flags, long refcon);

```

Pascal Summary**Constants**

```

CONST
    videoDigitizerComponentType = 'vdig'; {standard type for video }
                                         { digitizer components}

    {input format standards}
    ntscIn      = 0; {National Television System Committee}
    palIn       = 1; {Phase Alternation Line}
    secamIn     = 2; {Sequential Color with Memory}

    {input formats}
    compositeIn = 0; {no color separation of channels}
    sVideoIn   = 1; {s-video (Super VHS)}
    rgbComponentIn = 2; {separate channels for red, green, & blue}

```

Video Digitizer Components

```

{video digitizer PlayThru states}
vdPlayThruOff      = 0;  {playthrough off}
vdPlayThruOn       = 1;  {playthrough on}

{field preference options in VDGetFieldPreference function}
vdUseAnyField      = 0;  {digitizer component decides which field to use}
vdUseOddField      = 1;  {digitizer component uses odd field}
vdUseEvenField     = 2;  {digitizer component uses even field}

{input color space modes}
vdDigitizerBW      = 0;  {digitizer component uses black and white}
vdDigitizerRGB     = 1;  {digitizer component uses red, green, and blue}

{phase lock loop modes}
vdBroadcastMode    = 0;  {broadcast or laser disk video mode}
vdVTRMode          = 1;  {video cassette recorder or magnetic media mode}

{video digitizer component types}
vdTypeBasic        = 0;  {basic component does not support clipping}
vdTypeAlpha        = 1;  {component supports clipping with alpha channel}
vdTypeMask         = 2;  {component supports clipping with mask plane}
vdTypeKey          = 3;  {supports clipping with one or more key colors}

{digitizer input capability/current flags}
digiInDoesNTSC     = $1;   {digitizer supports NTSC input}
digiInDoesPAL      = $2;   {digitizer supports PAL input}
digiInDoesSECAM    = $4;   {digitizer supports SECAM input}
digiInDoesGenLock  = $80;  {digitizer supports genlock}
digiInDoesComposite = $100; {digitizer supports composite input type}
digiInDoesSVideo   = $200; {digitizer supports s-video input type}
digiInDoesComponent = $400; {digitizer supports component input type}
digiInVTR_Broadcast = $800; {digitizer can differentiate between }
                        { magnetic media & broadcast}
digiInDoesColor    = $1000; {digitizer supports color}
digiInDoesBW       = $2000; {digitizer supports black and white}

{digitizer input current flag (valid only during active operating )
{ conditions}}
digiInSignalLock   = $80000000; {digitizer detects input signal is }
                        { locked--this bit equals }
                        { horiz lock || vertical lock}

```

Video Digitizer Components

```

{digitizer output capability/current flags}
digiOutDoes1      = $1;      {digitizer supports 1-bit pixels}
digiOutDoes2      = $2;      {digitizer supports 2-bit pixels}
digiOutDoes4      = $4;      {digitizer supports 4-bit pixels}
digiOutDoes8      = $8;      {digitizer supports 8-bit pixels}
digiOutDoes16     = $10;     {digitizer supports 16-bit pixels}
digiOutDoes32     = $20;     {digitizer supports 32-bit pixels}
digiOutDoesDither = $40;     {digitizer dithers in indexed modes}
digiOutDoesStretch = $80;     {digitizer can arbitrarily stretch}
digiOutDoesShrink = $100;    {digitizer can arbitrarily shrink}
digiOutDoesMask   = $200;    {digitizer can mask to clipping }
                           { regions}

digiOutDoesDouble = $800;    {can stretch to exactly double size}
digiOutDoesQuad   = $1000;   {can stretch to exactly quadruple }
                           { size}

digiOutDoesQuarter = $2000;  {can shrink to exactly 1/4 size}
digiOutDoesSixteenth = $4000; {can shrink to exactly 1/16 size}
digiOutDoesRotate  = $8000;  {supports rotation transformations}
digiOutDoesHorizFlip = $10000; {supports horizontal flips Sx < 0}
digiOutDoesVertFlip = $20000; {supports vertical flips Sy < 0}
digiOutDoesSkew    = $40000; {supports skew (shear, twist)}
digiOutDoesBlend   = $80000; {digitizer performs blend operations}
digiOutDoesWarp    = $100000; {digitizer performs warp operations}
digiOutDoesHW_DMA  = $200000; {not constrained to logical device}
digiOutDoesHWPlayThru= $400000; {doesn't need time to play through}
digiOutDoesILUT    = $800000; {does lookup for index modes}
digiOutDoesKeyColor = $1000000; {performs key color functions too}
digiOutDoesAsyncGrabs= $2000000; {performs asynchronous grabs}
digiOutDoesUnreadableScreenBits
                   = $4000000; {playthru doesn't generate readable }
                           { bits on screen}

digiOutDoesCompress = $8000000; {supports compressed source devices}
digiOutDoesCompressOnly
                   = $10000000; {can't draw images}

digiOutDoesPlayThruDuringCompress
                   = $20000000; {can play while providing compressed }
                           { data}

{video digitizer interface}
kSelectVDGetMaxSrcRect      = $1; {VDGetMaxSrcRect (required)}
kSelectVDGetActiveSrcRect   = $2; {VDGetActiveSrcRect (required)}
kSelectVDSetDigitizerRect   = $3; {VDSetDigitizerRect (required)}
kSelectVDGetDigitizerRect   = $4; {VDGetDigitizerRect (required)}
kSelectVDGetVBlankRect      = $5; {VDGetVBlankRect (required)}

```

Video Digitizer Components

kSelectVDGetMaskPixMap	= \$6; {VDGetMaskPixMap}
kSelectVDGetPlayThruDestination	= \$8; {VDGetPlayThruDestination } { (required)}
kSelectVDUseThisCLUT	= \$9; {VDUseThisCLUT}
kSelectVDSetInputGammaValue	= \$A; {VDSetInputGammaValue}
kSelectVDGetInputGammaValue	= \$B; {VDGetInputGammaValue}
kSelectVDSetBrightness	= \$C; {VDSetBrightness}
kSelectVDGetBrightness	= \$D; {VDGetBrightness}
kSelectVDSetContrast	= \$E; {VDSetContrast}
kSelectVDSetHue	= \$F; {VDSetHue}
kSelectVDSetSharpness	= \$10; {VDSetSharpness}
kSelectVDSetSaturation	= \$11; {VDSetSaturation}
kSelectVDGetContrast	= \$12; {VDGetContrast}
kSelectVDGetHue	= \$13; {VDGetHue}
kSelectVDGetSharpness	= \$14; {VDGetSharpness}
kSelectVDGetSaturation	= \$15; {VDGetSaturation}
kSelectVDGrabOneFrame	= \$16; {VDGrabOneFrame (required)}
kSelectVDGetMaxAuxBuffer	= \$17; {VDGetMaxAuxBuffer}
kSelectVDGetDigitizerInfo	= \$19; {VDGetDigitizerInfo}
kSelectVDGetCurrentFlags	= \$1A; {VDGetCurrentFlags}
kSelectVDSetKeyColor	= \$1B; {VDSetKeyColor}
kSelectVDGetKeyColor	= \$1C; {VDGetKeyColor}
kSelectVDAAddKeyColor	= \$1D; {VDAAddKeyColor}
kSelectVDGetNextKeyColor	= \$1E; {VDGetNextKeyColor}
kSelectVDSetKeyColorRange	= \$1F; {VDSetKeyColorRange}
kSelectVDGetKeyColorRange	= \$20; {VDGetKeyColorRange}
kSelectVDSetDigitizerUserInterrupt	= \$21; {VDSetDigitizerUserInterrupt}
kSelectVDSetInputColorSpaceMode	= \$22; {VDSetInputColorSpaceMode}
kSelectVDGetInputColorSpaceMode	= \$23; {VDGetInputColorSpaceMode}
kSelectVDSetClipState	= \$24; {VDSetClipState}
kSelectVDGetClipState	= \$25; {VDGetClipState}
kSelectVDSetClipRgn	= \$26; {VDSetClipRgn}
kSelectVDClearClipRgn	= \$27; {VDClearClipRgn}
kSelectVDGetCLUTInUse	= \$28; {VDGetCLUTInUse}
kSelectVDSetPLLFilterType	= \$29; {VDSetPLLFilterType}
kSelectVDGetPLLFilterType	= \$2A; {VDGetPLLFilterType}
kSelectVDGetMaskandValue	= \$2B; {VDGetMaskandValue}
kSelectVDSetMasterBlendLevel	= \$2C; {VDSetMasterBlendLevel}
kSelectVDSetPlayThruDestination	= \$2D; {VDSetPlayThruDestination}
kSelectVDSetPlayThruOnOff	= \$2E; {VDSetPlayThruOnOff}
kSelectVDSetFieldPreference	= \$2F; {VDSetFieldPreference } { (required)}

Video Digitizer Components

Video Digitizer Components

Data Types

TYPE

```

VideoDigitizerComponent      = ComponentInstance; {video digitizer }
                                { component}
VideoDigitizerError          = ComponentResult;   {video digitizer }
                                { error}
VdigIntProc                  = ComponentResult;

```

```
DigitizerInfo =
```

RECORD

```

    vdigType:                Integer; {type of digitizer component}
    inputCapabilityFlags:    LongInt; {input video signal features}
    outputCapabilityFlags:   LongInt; {output digitized video data features}
    inputCurrentFlags:       LongInt; {status of input video signal}
    outputCurrentFlags:      LongInt; {status of output digitized data}
    slot:                    Integer; {temporary for connection purposes}
    gdh:                     GDHandle; {temporary for digitizers with }
                                { preferred screen}
    maskgdh:                  GDHandle; {temporary for digitizers }
                                { with mask planes}
    minDestHeight:            Integer; {smallest resizable height}
    minDestWidth:             Integer; {smallest resizable width}
    maxDestHeight:            Integer; {largest resizable height}
    maxDestWidth:             Integer; {largest resizable width}
    blendLevels:              Integer; {number of blend levels supported (2 }
                                { if 1 bit mask)}
    Private:                  LongInt; {reserved--set to 0}
END;

```

```
VdigBufferRec =
```

RECORD

```

    dest:      PixMapHandle; {handle to pixel map for destination buffer}
    location:   Point;        {location of video destination in pixel map}
    reserved:   LongInt;      {reserved--set to 0}
END;

```

```
VdigBufferRecListPtr      = ^VdigBufferRecList;
```

```
VdigBufferRecListHandle   = ^VdigBufferRecListPtr;
```

```
VdigBufferRecList =
```

RECORD

```

    count:      Integer;          {buffers defined by this record}
    matrix:      MatrixRecordPtr ; {transformation matrix applied to }
                                { dest rect before image displayed}

```

Video Digitizer Components

```

mask:          RgnHandle;          {clip rgn applied to dest rect }
                                   { before image displayed}

list:          ARRAY[0..0] OF VdigBufferRec;
                                   {array of output buffer specs}

END;

VDCompressionListHandle = ^VDCompressionListPtr;
VDCompressionList =
  RECORD
    codec:      CodecComponent;    {component ID}
    cType:      CodecType;         {compressor type}
    typeName:   Str63;             {compression algorithm}
    name:       Str63;             {compressor name string}
    formatFlags: LongInt;          {data format flags}
    compressFlags: LongInt;        {capabilities flags}
    reserved:   LongInt;           {set to 0}
  END;

```

Video Digitizer Component Routines

Getting Information About Video Digitizer Components

```

FUNCTION VDGetDigitizerInfo
  (ci: VideoDigitizerComponent;
   VAR info: DigitizerInfo): VideoDigitizerError;

FUNCTION VDGetCurrentFlags
  (ci: VideoDigitizerComponent;
   VAR inputCurrentFlag: LongInt;
   VAR outputCurrentFlag: LongInt):
  VideoDigitizerError;

```

Setting Source Characteristics

```

FUNCTION VDGetMaxSrcRect
  (ci: VideoDigitizerComponent;
   VAR maxSrcRect: Rect): VideoDigitizerError;

FUNCTION VDGetActiveSrcRect
  (ci: VideoDigitizerComponent; inputStd: Integer;
   VAR activeSrcRect: Rect): VideoDigitizerError;

FUNCTION VDGetVBlankRect
  (ci: VideoDigitizerComponent;
   VAR vBlankRect: Rect): VideoDigitizerError;

FUNCTION VDSetDigitizerRect
  (ci: VideoDigitizerComponent;
   VAR digitizerRect: Rect): VideoDigitizerError;

```

Video Digitizer Components

```

FUNCTION VDGetDigitizerRect
    (ci: VideoDigitizerComponent;
     VAR digitizerRect: Rect): VideoDigitizerError;

```

Selecting an Input Source

```

FUNCTION VDGetNumberOfInputs
    (ci: VideoDigitizerComponent;
     VAR inputs: Integer): VideoDigitizerError;

FUNCTION VDSetInput
    (ci: VideoDigitizerComponent;
     input: Integer): VideoDigitizerError;

FUNCTION VDGetInput
    (ci: VideoDigitizerComponent;
     VAR input: Integer): VideoDigitizerError;

FUNCTION VDGetInputFormat
    (ci: VideoDigitizerComponent; input: Integer;
     VAR format: Integer): VideoDigitizerError;

FUNCTION VDSetInputStandard
    (ci: VideoDigitizerComponent;
     inputStandard: Integer): VideoDigitizerError;

```

Setting Video Destinations

```

FUNCTION VDSetPlayThruDestination
    (ci: VideoDigitizerComponent;
     dest: PixMapHandle; VAR destRect: Rect;
     VAR m: MatrixRecord;
     mask: RgnHandle): VideoDigitizerError;

FUNCTION VDPreflightDestination
    (ci: VideoDigitizerComponent;
     VAR digitizerRect: Rect; dest: PixMapHandle;
     VAR destRect: Rect;
     VAR m: MatrixRecord): VideoDigitizerError;

FUNCTION VDGetPlayThruDestination
    (ci: VideoDigitizerComponent;
     VAR dest: PixMapHandle; VAR destRect: Rect;
     VAR m: MatrixRecord;
     VAR mask: RgnHandle): VideoDigitizerError;

FUNCTION VDSetPlayThruGlobalRect
    (ci: VideoDigitizerComponent;
     theWindow: GrafPtr; VAR globalRect: Rect):
     VideoDigitizerError;

FUNCTION VDPreflightGlobalRect
    (ci: VideoDigitizerComponent;
     theWindow: GrafPtr; VAR globalRect: Rect):
     VideoDigitizerError;

```

Video Digitizer Components

```

FUNCTION VDGetMaxAuxBuffer (ci: VideoDigitizerComponent;
                           VAR pm: PixMapHandle; VAR r: Rect):
                           VideoDigitizerError;

```

Controlling Compressed Source Devices

```

FUNCTION VDGetCompressionTypes
                           (ci: VideoDigitizerComponent;
                           h: VDCompressionListHandle):
                           VideoDigitizerError;

FUNCTION VDSetCompression (ci: VideoDigitizerComponent;
                           compressType: OSType; depth: Integer;
                           VAR bounds: Rect; spatialQuality: CodecQ;
                           temporalQuality: CodecQ;
                           keyFrameRate: LongInt): VideoDigitizerError;

FUNCTION VDSetCompressionOnOff
                           (ci: VideoDigitizerComponent; state: Boolean):
                           VideoDigitizerError;

FUNCTION VDGrabOneFrameAsync
                           (ci: VideoDigitizerComponent; buffer: Integer):
                           VideoDigitizerError;

FUNCTION VDCompressDone   (ci: VideoDigitizerComponent;
                           VAR done: Boolean; VAR theData: Ptr;
                           VAR dataSize: LongInt; VAR similarity: Byte;
                           VAR t: TimeRecord): VideoDigitizerError;

FUNCTION VDReleaseCompressBuffer
                           (ci: VideoDigitizerComponent;
                           bufferAddr: Ptr): VideoDigitizerError;

FUNCTION VDGetImageDescription
                           (ci: VideoDigitizerComponent;
                           desc: ImageDescriptionHandle):
                           VideoDigitizerError;

FUNCTION VDResetCompressSequence
                           (ci: VideoDigitizerComponent):
                           VideoDigitizerError;

FUNCTION VDSetTimeBase    (ci: VideoDigitizerComponent; t: TimeBase):
                           VideoDigitizerError;

```

Controlling Digitization

```

FUNCTION VDSetPlayThruOnOff (ci: VideoDigitizerComponent;
                             state: Integer): VideoDigitizerError;

FUNCTION VDGrabOneFrame     (ci: VideoDigitizerComponent):
                             VideoDigitizerError;

```

Video Digitizer Components

```

FUNCTION VDSaveBuffers      (ci: VideoDigitizerComponent;
                           bufferList: VdigBufferRecListHandle):
                           VideoDigitizerError;

FUNCTION VDReleaseAsyncBuffers
                           (ci: VideoDigitizerComponent):
                           VideoDigitizerError;

FUNCTION VDGrabOneFrameAsync
                           (ci: VideoDigitizerComponent;
                           nextBuffer: Integer): VideoDigitizerError;

FUNCTION VDDone             (ci: VideoDigitizerComponent,
                           buffer: Integer): LongInt;

FUNCTION VDSaveFrameRate   (ci: VideoDigitizerComponent;
                           framesPerSecond: Fixed): VideoDigitizerError;

FUNCTION VDGetDataRate      (ci: VideoDigitizerComponent;
                           VAR milliSecPerFrame: LongInt;
                           VAR framesPerSecond: Fixed;
                           VAR bytesPerSecond: LongInt):
                           VideoDigitizerError;

```

Controlling Color

```

FUNCTION VDUseThisCLUT      (ci: VideoDigitizerComponent;
                           colorTableHandle: CTabHandle):
                           VideoDigitizerError;

FUNCTION VDGetCLUTInUse     (ci: VideoDigitizerComponent;
                           VAR colorTableHandle: CTabHandle):
                           VideoDigitizerError;

FUNCTION VDSetInputColorSpaceMode
                           (ci: VideoDigitizerComponent;
                           colorSpaceMode: Integer): VideoDigitizerError;

FUNCTION VDGetInputColorSpaceMode
                           (ci: VideoDigitizerComponent;
                           VAR colorSpaceMode: Integer):
                           VideoDigitizerError;

FUNCTION VDGetDMADepths     (ci: VideoDigitizerComponent;
                           VAR depthArray: LongInt; VAR preferredDepth:
                           LongInt): VideoDigitizerError;

```

Controlling Analog Video

```

FUNCTION VGetVideoDefaults (ci: VideoDigitizerComponent;
    VAR blackLevel: Integer;
    VAR whiteLevel: Integer;
    VAR brightness: Integer; VAR hue: Integer;
    VAR saturation: Integer; VAR contrast: Integer;
    VAR sharpness: Integer): VideoDigitizerError;

FUNCTION VSetBlackLevelValue
    (ci: VideoDigitizerComponent;
    VAR blackLevel: Integer): VideoDigitizerError;

FUNCTION VGetBlackLevelValue
    (ci: VideoDigitizerComponent;
    VAR blackLevel: Integer): VideoDigitizerError;

FUNCTION VSetWhiteLevelValue
    (ci: VideoDigitizerComponent;
    VAR whiteLevel: Integer): VideoDigitizerError;

FUNCTION VGetWhiteLevelValue
    (ci: VideoDigitizerComponent;
    VAR whiteLevel: Integer): VideoDigitizerError;

FUNCTION VSetHue
    (ci: VideoDigitizerComponent;
    VAR hue: Integer): VideoDigitizerError;

FUNCTION VGetHue
    (ci: VideoDigitizerComponent;
    VAR hue: Integer): VideoDigitizerError;

FUNCTION VSetSaturation
    (ci: VideoDigitizerComponent;
    VAR saturation: Integer): VideoDigitizerError;

FUNCTION VGetSaturation
    (ci: VideoDigitizerComponent;
    VAR saturation: Integer): VideoDigitizerError;

FUNCTION VSetBrightness
    (ci: VideoDigitizerComponent;
    VAR brightness: Integer): VideoDigitizerError;

FUNCTION VGetBrightness
    (ci: VideoDigitizerComponent;
    VAR brightness: Integer): VideoDigitizerError;

FUNCTION VSetContrast
    (ci: VideoDigitizerComponent;
    VAR contrast: Integer): VideoDigitizerError;

FUNCTION VGetContrast
    (ci: VideoDigitizerComponent;
    VAR contrast: Integer): VideoDigitizerError;

FUNCTION VSetSharpness
    (ci: VideoDigitizerComponent;
    VAR sharpness: Integer): VideoDigitizerError;

FUNCTION VGetSharpness
    (ci: VideoDigitizerComponent;
    VAR sharpness: Integer): VideoDigitizerError;

FUNCTION VSetInputGammaRecord
    (ci: VideoDigitizerComponent;
    inputGammaPtr: VDGamRecPtr):
    VideoDigitizerError;

```

Video Digitizer Components

```

FUNCTION VDGetInputGammaRecord
    (ci: VideoDigitizerComponent;
     VAR inputGammaPtr: VDGamRecPtr):
    VideoDigitizerError;

FUNCTION VDSetInputGammaValue
    (ci: VideoDigitizerComponent; channel1: Fixed;
     channel2: Fixed; channel3: Fixed):
    VideoDigitizerError;

FUNCTION VDGetInputGammaValue
    (ci: VideoDigitizerComponent;
     VAR channel1: Fixed; VAR channel2: Fixed;
     VAR channel3: Fixed): VideoDigitizerError;

```

Selectively Displaying Video

```

FUNCTION VDSetKeyColor    (ci: VideoDigitizerComponent;
                           index: LongInt): VideoDigitizerError;

FUNCTION VDGetKeyColor    (ci: VideoDigitizerComponent;
                           VAR index: LongInt): VideoDigitizerError;

FUNCTION VDSetKeyColorRange
    (ci: VideoDigitizerComponent;
     VAR minRGB: RGBColor; VAR maxRGB: RGBColor):
    VideoDigitizerError;

FUNCTION VAddKeyColor     (ci: VideoDigitizerComponent;
                           VAR index: LongInt): VideoDigitizerError;

FUNCTION VDGetKeyColorRange
    (ci: VideoDigitizerComponent;
     VAR minRGB: RGBColor; VAR maxRGB: RGBColor):
    VideoDigitizerError;

FUNCTION VDGetNextKeyColor (ci: VideoDigitizerComponent;
                            index: LongInt): VideoDigitizerError;

FUNCTION VDSetMasterBlendLevel
    (ci: VideoDigitizerComponent;
     VAR blendLevel: Integer): VideoDigitizerError;

FUNCTION VDGetMaskandValue (ci: VideoDigitizerComponent;
                            blendLevel: Integer; VAR mask: LongInt;
                            VAR value: LongInt): VideoDigitizerError;

FUNCTION VDGetMaskPixMap  (ci: VideoDigitizerComponent;
                            maskPixMap: PixMapHandle): VideoDigitizerError;

```


Clipping

```

FUNCTION VDSaveClipRgn      (ci: VideoDigitizerComponent;
                             clipRegion: RgnHandle): VideoDigitizerError;

FUNCTION VDClearClipRgn    (ci: VideoDigitizerComponent;
                             clipRegion: RgnHandle): VideoDigitizerError;

FUNCTION VDSetClipState    (ci: VideoDigitizerComponent;
                             clipEnable: Integer): VideoDigitizerError;

FUNCTION VDGetClipState    (ci: VideoDigitizerComponent;
                             VAR clipEnable: Integer): VideoDigitizerError;

```

Utility Functions

```

FUNCTION VDSetPLLFilterType (ci: VideoDigitizerComponent;
                             pllType: Integer): VideoDigitizerError;

FUNCTION VDGetPLLFilterType (ci: VideoDigitizerComponent;
                             VAR pllType: Integer): VideoDigitizerError;

FUNCTION VDSetFieldPreference (ci: VideoDigitizerComponent;
                                fieldFlag: Integer): VideoDigitizerError;

FUNCTION VDGetFieldPreference (ci: VideoDigitizerComponent;
                                VAR fieldFlag: Integer): VideoDigitizerError;

FUNCTION VDSetDigitizerUserInterrupt (ci: VideoDigitizerComponent;
                                       flags: LongInt; userInterruptProc: ProcPtr;
                                       refcon: LongInt): VideoDigitizerError;

FUNCTION VDGetSoundInputDriver (ci: VideoDigitizerComponent;
                                soundDriverName: Str255): VideoDigitizerError;

FUNCTION VDGetPreferredTimeScale (ci: VideoDigitizerComponent;
                                   preferred: TimeScale): VideoDigitizerError;

```

Application-Defined Routine

```

PROCEDURE MyInterruptProc (flags: LongInt; refcon: LongInt);

```

Result Codes

Video Digitizer Components