

This chapter discusses derived media handler components. **Derived media handler components** allow the Movie Toolbox to play the data in a media. These components isolate the Movie Toolbox from the details of how or where a particular media is stored. This not only frees the Movie Toolbox from reading and writing media data, but also makes QuickTime extensible to new data formats.

These components are referred to as *derived* components because they rely on the services of a common base media handler component, which is supplied by Apple. The **base media handler component** handles most of the duties that must be performed by all media handlers. Your derived media handler component extends the services provided by the base media handler.

This chapter is divided into the following sections:

- “About Derived Media Handler Components” provides a general introduction to components of this type.
- “Creating a Derived Media Handler Component” provides a sample program for the implementation of such a component for PICT files.
- “pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /* remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived Media Handler Components Reference” presents detailed information about the functions that are supported by these components.
- “Summary of Derived Media Handler Components” contains a condensed listing of the constants, data structures, and functions supported by these components.

This chapter addresses developers of derived media handler components. You should never need to use the facilities of a derived media handler directly—only the Movie Toolbox calls derived media handler components. The functions described in this chapter define the functional interface that your component must support.

As components, derived media handlers rely on the facilities of the Component Manager. To use any component, your application must also use the Component Manager. If you are not familiar with this manager, see the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox*. In addition, you should be familiar with the Movie Toolbox in general and the concept of media structures in particular. See the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for more information.

Note

Throughout this chapter, the terms *media handler* and *handler* refer to media handler components. Apple’s sound and video handlers are not derived media handlers, so you cannot override them using the functions described in this chapter. Apple’s text media handler, on the other hand, is built on the base media handler. ♦

About Derived Media Handler Components

This section provides background information about media handler components in general and derived media handler components in particular. After reading this section, you should understand why media handler components exist and whether you need to create a derived media handler component.

Media Handler Components

Media handler components allow the Movie Toolbox to play a movie's data. The Movie Toolbox, by itself, cannot read or write movie data. Rather, media handlers perform input and output services on behalf of the Movie Toolbox. The Movie Toolbox gains access to the appropriate media handler for a particular movie track by examining the track's media. That data structure identifies the media handler that created and maintains the media (see the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime* for more information about the relationship between a movie, its tracks, and each track's media).

Each media handler is primarily responsible for understanding the format and content of the media type it supports. The media handler is intimately familiar with the sample structure used in its media, the compression techniques used to store the media's sample data, and the performance characteristics of the device that stores the media.

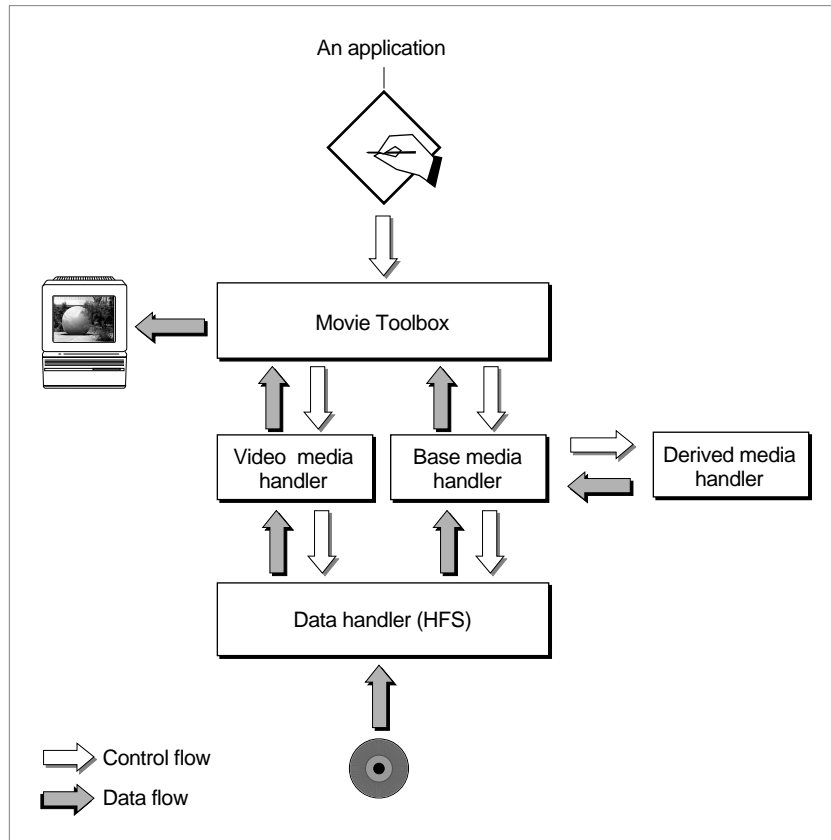
During movie playback, the media handler draws its media's data on the screen and plays the media's sounds. The media handler may use the services of other managers such as the Image Compression Manager for compressed image data and the Sound Manager for sound data. When an application creates a movie, media handlers store the movie's data. The actual reading and writing of media data are performed by another component, the **data handler**. For details on the Image Compression Manager, see *Inside Macintosh: QuickTime*. For more on the Sound Manager, see *Inside Macintosh: More Macintosh Toolbox*.

Applications never directly use the services of media handlers. The Movie Toolbox controls all movie data storage and retrieval on behalf of QuickTime applications.

Derived Media Handler Components

Figure 10-1 shows the logical relationships between applications, the Movie Toolbox, media handlers, and data handlers.

Figure 10-1 Logical relationships between the Movie Toolbox and media handlers



Apple had three primary goals for isolating the Movie Toolbox and QuickTime applications from the details of media data access. First, the isolation allows programmers who develop the Movie Toolbox and QuickTime applications to focus on the specifics of the problems they are addressing, freed from concerns about data access. Second, this architecture allows QuickTime to be easily extended to accommodate new storage devices and technologies. Third, by documenting the media handler interface, developers can create their own, special-purpose media handlers that work with QuickTime.

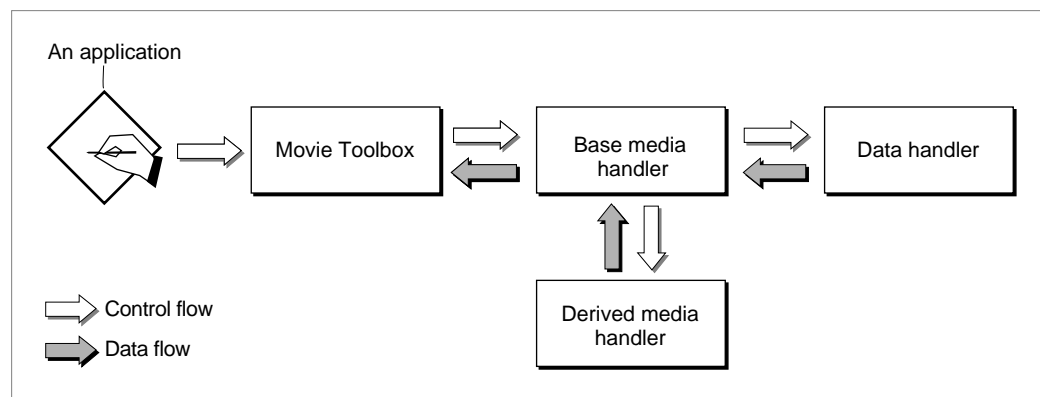
Derived Media Handler Components

Much of what a media handler component must do is common to all media handlers. Managing a connection with the appropriate data handler, retrieving movie data from media samples, and storing movie data into new samples account for a substantial part of every media handler's responsibilities. To make it easier for developers to create media handler components, Apple provides a base media handler component that performs most of the common duties of a media handler.

Apple's base media handler component eliminates much of the work you would have to do to create your own media handler component. The base media handler interacts with both the Movie Toolbox and the appropriate data handler, so that your media handler only has to deal with service requests, and you can ignore many of the housekeeping functions. It understands the format of Apple's media samples and sample descriptions, so that your media handler only has to worry about the actual media data. Finally, it provides basic services that your media handler can use to accommodate unusual display environments.

When you build your media handler component on top of the base media handler, your media handler is known as a *derived media handler component*. This terminology is borrowed from object-oriented development and refers to the fact that your media handler is based on, or derived from, the services provided by Apple's base media handler. Figure 10-2 shows the relationship between the base media handler, derived media handlers, the Movie Toolbox, and data handler components.

Figure 10-2 Relationship between the base media handler component and derived media handlers



You should consider deriving your media handler from Apple's base media handler component if your media requires low to moderate data throughput. Apple's base media handler can support data rates up to 32 kilobits per second. This rate is adequate for such data types as text, sound effects, animation, annotations, or MIDI (Musical

Instrument Digital Interface) sound data. However, Apple's base media handler is not appropriate for CD-quality sound, which may require data rates of up to 176 kilobits per second.

Creating a Derived Media Handler Component

This section provides an example of creating a derived media handler component. The functional interface that your derived media handler component must support is described in “pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /* remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived Media Handler Components Reference” beginning on page 10-15.

Before reading this section, you should be familiar with how to create components. See the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox* for a complete discussion of components—how to use them and how to create them.

Apple has defined a component type value for media handler components. All components of this type have the same type value. You can use the following constant to specify this component type:

```
#define MediaHandlerType 'mhlr' /* media handler */
```

Apple has defined a functional interface for derived media handler components. For information about the functions that your component must support, see “pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /* remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived Media Handler Components Reference” beginning on page 10-15. You can use the following constants to refer to the request codes for each of the functions that your component must support:

```
enum {
    kMediaInitializeSelect          = 0x501, /* MediaInitialize */
    kMediaSetHandlerCapabilitiesSelect = 0x502,
                                     /* MediaSetHandlerCapabilities */
    kMediaIdleSelect                = 0x503, /* MediaIdle */
    kMediaGetMediaInfoSelect         = 0x504, /* MediaGetMediaInfo */
    kMediaPutMediaInfoSelect         = 0x505, /* MediaPutMediaInfo */
    kMediaSetActiveSelect           = 0x506, /* MediaSetActive */
    kMediaSetRateSelect             = 0x507, /* MediaSetRate */
    kMediaGGetStatusSelect          = 0x508, /* MediaGGetStatus */
    kMediaTrackEditedSelect         = 0x509, /* MediaTrackEdited */
    kMediaSetMediaTimeScaleSelect   = 0x50A, /* MediaSetMediaTimeScale */
    kMediaSetMovieTimeScaleSelect   = 0x50B, /* MediaSetMovieTimeScale */
    kMediaSetGWorldSelect           = 0x50C, /* MediaSetGWorld */
}
```

Derived Media Handler Components

```

kMediaSetDimensionsSelect      = 0x50D, /* MediaSetDimensions */
kMediaSetClipSelect           = 0x50E, /* MediaSetClip */
kMediaSetMatrixSelect         = 0x50F, /* MediaSetMatrix */
kMediaGetTrackOpaqueSelect     = 0x510, /* MediaGetTrackOpaque */
kMediaSetGraphicsModeSelect    = 0x511, /* MediaSetGraphicsMode */
kMediaGetGraphicsModeSelect    = 0x512, /* MediaGetGraphicsMode */
kMediaGSetVolumeSelect        = 0x513, /* MediaGSetVolume */
kMediaSetSoundBalanceSelect    = 0x514, /* MediaSetSoundBalance */
kMediaGetSoundBalanceSelect    = 0x515, /* MediaGetSoundBalance */
kMediaGetNextBoundsChangeSelect = 0x516,
                                /* MediaGetNextBoundsChange */
kMediaGetSrcRgnSelect          = 0x517, /* MediaGetSrcRgn */
kMediaPrerollSelect            = 0x518, /* MediaPreroll */
kMediaSampleDescriptionChangedSelect = 0x519,
                                /* MediaSampleDescriptionChanged */
kMediaHasCharacteristicSelect  = 0x51A  /* MediaHasCharacteristic */
};

```

Component Flags for Derived Media Handlers

The Component Manager allows you to specify information about your component's capabilities in the `componentFlags` field of the component description record. You must set this component flag to 1 in the component description that is associated with your derived media handler:

`mediaHandlerFlagBaseClient`

Indicates that your component is derived from another component.

Setting this flag to 1 tells the Component Manager that your component is a client of the base media handler.

Request Processing

Because your derived media handler is based on the base media handler component, you avoid many of the details involved in creating a media handler. However, your derived media handler must observe a few rules when processing service requests.

These rules are as follows:

- When you receive an open request from the Component Manager, in addition to the other processing you perform on your own behalf, you must also open a connection to the base media handler component. You should save the component instance that is returned by the Component Manager so that your media handler can use the services of the base media handler.
- The base media handler has a component type of `MediaHandlerType` (which is set to 'mhlr') and a component subtype of `BaseMediaType` (which is set to 'gnrc'). You can use these values with the Component Manager's `OpenDefaultComponent` function to open a connection to the base media handler.

- At this time, you must also tell the base media handler that your handler is derived from it. Use the Component Manager's `OpenComponent` function to create a component instance of your media handler as a descendant of the base media handler. After calling that function, you should send the `kComponentSetTargetSelect` request to the base media handler, so that it knows your media handler is derived from it. Use the Component Manager's `ComponentSetTarget` function to send a target request.
- When you receive a close request from the Component Manager, be sure to close your handler's connection to the base media handler component. Use the Component Manager's `CloseComponent` function.
- Your derived media handler must support the target request, so that your component can be used by other media handlers.
- Be sure to pass all unsupported service requests to the base media handler component. Use the Component Manager's `DelegateComponentCall` function to pass these requests to the base media handler.
- If your media handler component competes for potentially scarce system resources, your component should release those resources when you aren't using them. For example, if you are creating a media handler that uses sound, you might use sound channels. Because there are a limited number of sound channels available, your component should free its channels whenever your media is not playing or has been stopped. You can reallocate the channels when you start playing or your component's `MediaPreroll` function is called.

A Sample Derived Media Handler Component

This section supplies a sample program that implements a derived media handler component for PICT images.

Implementing the Required Component Functions

Listing 10-1 supplies the component dispatchers for the media handler component for PICT images together with the required functions.

Listing 10-1 Implementing the required functions

```
typedef struct {
    ComponentInstance self;
    ComponentInstance parent;
    ComponentInstance delegateComponent;
    Fixed              width;
    Fixed              height;
    MatrixRecord       matrix;
    Media              media;
    Track              track;
} PictGlobalsRecord, *PictGlobals;
```

Derived Media Handler Components

```

pascal ComponentResult PictMediaDispatch
    (ComponentParameters *params,
     Handle storage)
{
    OSErr err = badComponentSelector;
    ComponentFunction componentProc = 0;

    switch (params->what) {
        case kComponentOpenSelect:
            componentProc = PictOpen; break;
        case kComponentCloseSelect:
            componentProc = PictClose; break;
        case kComponentCanDoSelect:
            componentProc = PictCanDo; break;
        case kComponentVersionSelect:
            componentProc = PictVersion; break;
        case kComponentTargetSelect:
            componentProc = PictVersion; break;
        case kMediaInitializeSelect:
            componentProc = PictInitialize; break;
        case kMediaIdleSelect:
            componentProc = PictIdle; break;
        case kMediaSetDimensionsSelect:
            componentProc = PictSetDimensions; break;
        case kMediaSetMatrixSelect:
            componentProc = PictSetMatrix; break;
    }
    if (componentProc)
        err = CallComponentFunctionWithStorage (storage, params,
                                                componentProc);
    else
        err = DelegateComponentCall (params, ((PictGlobals)
                                                storage)->delegateComponent);

    return err;
}

pascal ComponentResult PictCanDo (PictGlobals globals,
    short ftnNumber)
{
    switch (ftnNumber) {
        case kComponentOpenSelect:
        case kComponentCloseSelect:

```


Derived Media Handler Components

```

        case kComponentCanDoSelect:
        case kComponentVersionSelect:
        case kComponentTargetSelect:

        case kMediaInitializeSelect:
        case kMediaIdleSelect:
        case kMediaSetDimensionsSelect:
        case kMediaSetMatrixSelect:
            return true;
        default:
            return ComponentFunctionImplemented
                (globals->delegateComponent, ftnNumber);
    }
}

pascal ComponentResult PictVersion (PictGlobals globals)
{
    return 0x00020001;
}

pascal ComponentResult PictOpen(PictGlobals globals,
                                ComponentInstance self)
{
    OSErr err;

    /* allocate storage */
    globals = (PictGlobals)NewPtrClear(sizeof(PictGlobalsRecord));
    if (err = MemError()) return err;
    SetComponentInstanceStorage(self, (Handle)globals);
    globals->self = self;
    globals->parent = self;

    /* find a base media handler to serve as a delegate */
    globals->delegateComponent =
        OpenDefaultComponent (MediaHandlerType,
                              BaseMediaType);

    if (globals->delegateComponent)
        PictTarget(globals, self); /* set up the calling chain */
    else {
        DisposePtr((Ptr)globals);
        err = cantOpenHandler;
    }
    return err;
}

```

Derived Media Handler Components

```

pascal ComponentResult PictClose (PictGlobals globals,
                                   ComponentInstance self)
{
    if (globals) {
        if (globals->delegateComponent)
            CloseComponent(globals->delegateComponent);

        DisposePtr((Ptr)globals);
    }

    return noErr;
}

pascal ComponentResult PictTarget(PictGlobals store,
                                   ComponentInstance parentComponent)
{
    /* remember who is at the top of your calling chain */
    store->parent = parentComponent;

    /* and inform your delegate component of the change */
    ComponentSetTarget(store->delegateComponent, parentComponent);

    return noErr;
}

```

Initializing a Derived Media Handler Component

The derived media handler component is initialized by the Movie Toolbox's calling of the `MediaInitialize` function (described on page 10-18). You should then report the derived media handler capabilities to the base media handler before the Movie Toolbox starts working with your media by calling the `MediaSetHandlerCapabilities` function (described on page 10-38) from your `MediaInitialize` function.

Listing 10-2 is the initialization function for a derived media handler. The `PictInitialize` function stores the initial height, width, track movie matrix, media, and track of the derived media handler component. From `PictInitialize`, the `MediaSetHandlerCapabilities` function is called to inform the base media handler of its existence and features.

Listing 10-2 Initializing a derived media handler

```

pascal ComponentResult PictInitialize (PictGlobals store,
                                      GetMovieCompleteParams *gmc)
{
    /* remember some useful parameters */
    store->width = gmc->width;
    store->height = gmc->height;
    store->matrix = gmc->trackMovieMatrix;
    store->media = gmc->theMedia;
    store->track = gmc->theTrack;

    /* tell the base media handler about your derived
       media handler */
    MediaSetHandlerCapabilities(store->delegateComponent,
                               handlerHasSpatial, handlerHasSpatial);

    return noErr;
}

```

Drawing the Media Sample

The Movie Toolbox provides processing time to your derived media handler to display samples by calling the `MediaIdle` function (described on page 10-19). Your media handler may use this time to play its media sample. The code in Listing 10-3 allows the derived media handler component to draw the current media sample (in this case, a PICT image).

Listing 10-3 Drawing the media sample

```

pascal ComponentResult PictIdle (PictGlobals store,
                                TimeValue atMediaTime,
                                long flagsIn, long *flagsOut,
                                const TimeValue *tr)
{
    OSErr err;
    Rect r;
    Handle sample = NewHandle (0);

    if (err = MemError()) goto bail;
}

```

Derived Media Handler Components

```

    /* get the current sample */
    err = GetMediaSample (store->media, sample, 0, nil,
                        atMediaTime, nil, 0, 0, 0, 0, 0, 0);
    if (err) goto bail;

    /* draw it using the current matrix */
    SetRect (&r, 0, 0, FixRound (store->width),
            FixRound (store->height));
    TransformRect (&store->matrix, &r, nil);
    EraseRect (&r);
    DrawPicture ((PicHandle)sample, &r);

bail:
    DisposeHandle (sample);
    *flagsOut |= mDidDraw; /* let Movie Toolbox know you drew
                        something */

    return err;
}

pascal ComponentResult PictSetDimensions (PictGlobals store,
                                          Fixed width,
                                          Fixed height)
{
    /* remember the new track */
    store->width = width;
    store->height = height;
    return noErr;
}

pascal ComponentResult PictSetMatrix (PictGlobals store,
                                       MatrixRecord *trackMovieMatrix)
{
    /* remember the new display matrix */
    store->matrix = *trackMovieMatrix;
    return noErr;
}

```

return noErr;

}Derived Media Handler Components Reference

This section describes the functions that your derived media handler may support and the data structure that your component may use to interact with the base media handler.

Data Type

The `GetMovieCompleteParams` data type defines the layout of the complete movie parameter structure used by the `MediaInitialize` function (described on page 10-18):

```
typedef struct {
    short          version;           /* version; always 0 */
    Movie          theMovie;          /* movie identifier */
    Track          theTrack;          /* track identifier */
    Media          theMedia;          /* media identifier */
    TimeScale      movieScale;        /* movie's time scale */
    TimeScale      mediaScale;        /* media's time scale */
    TimeValue      movieDuration;     /* movie's duration */
    TimeValue      trackDuration;     /* track's duration */
    TimeValue      mediaDuration;     /* media's duration */
    Fixed          effectiveRate;     /* media's effective rate */
    TimeBase       timeBase;          /* media's time base */
    short          volume;            /* media's volume */
    Fixed          width;             /* width of display area */
    Fixed          height;            /* height of display area */
    MatrixRecord   trackMovieMatrix; /* transformation matrix */
    CGrafPtr       moviePort;         /* movie's graphics port */
    GDHandle       movieGD;           /* movie's graphics device */
    PixMapHandle   trackMatte;        /* track's matte */
} GetMovieCompleteParams;
```

Field descriptions

version	Specifies the version of this structure. This field is always set to 0.
theMovie	Identifies the movie that contains the current media's track. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the movie that is using your media.
theTrack	Identifies the track that contains the current media. This track identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the track that contains your media. For example, you might call the Movie Toolbox's <code>GetTrackNextInterestingTime</code> function in order to examine the track's edit list.

Derived Media Handler Components

<code>theMedia</code>	Identifies the current media. This media identifier is supplied by the Movie Toolbox. Your derived media handler can use this identifier to read samples or sample descriptions from the current media, using the Movie Toolbox's <code>GetMediaSample</code> and <code>GetMediaSampleDescription</code> functions (see <i>Inside Macintosh: QuickTime</i> for information about the Movie Toolbox).
<code>movieScale</code>	Specifies the time scale of the movie that contains the current media's track. If the Movie Toolbox changes the movie's time scale, the toolbox calls your derived media handler's <code>MediaSetMovieTimeScale</code> function, which is described on page 10-29.
<code>mediaScale</code>	Specifies the time scale of the current media. If the Movie Toolbox changes your media's time scale, the toolbox calls your derived media handler's <code>MediaSetMediaTimeScale</code> function, which is described on page 10-29.
<code>movieDuration</code>	Contains the movie's duration. This value is expressed in the movie's time scale.
<code>trackDuration</code>	Contains the track's duration. This value is expressed in the movie's time scale.
<code>mediaDuration</code>	Contains the media's duration. This value is expressed in the media's time scale.
<code>effectiveRate</code>	<p>Contains the media's effective rate. This rate ties the media's time scale to the passage of absolute time, and does not necessarily correspond to the movie's rate. This value takes into account any master time bases that may be serving the media's time base. The value of this field indicates the number of time units (in the media's time scale) that pass each second.</p> <p>This rate is represented as a 32-bit, fixed-point number. The high-order 16 bits contain the integer portion, and the low-order 16 bits contain the fractional portion. The rate is negative when time is moving backward for the media.</p> <p>Whenever the Movie Toolbox changes your media's effective rate, it calls your derived media handler's <code>MediaSetRate</code> function, which is discussed on page 10-26.</p>
<code>timeBase</code>	Identifies the media's time base.
<code>volume</code>	<p>Contains the media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.</p> <p>If the Movie Toolbox changes your media's volume, it calls your derived media handler's <code>MediaGSetVolume</code> function, which is discussed on page 10-37.</p>
<code>width</code>	Indicates the width, in pixels, of the track rectangle. This field, along with the <code>height</code> field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This

	<p>value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.</p> <p>If the Movie Toolbox modifies this rectangle, the toolbox calls your derived media handler's <code>MediaSetDimensions</code> function, which is discussed on page 10-32.</p> <p>Note that your media need not present only a rectangular image. The Movie Toolbox can use a clipping region to cause your media's image to be displayed in a region of arbitrary shape, and it can use a matte to control the image's transparency. The toolbox calls your derived media handler's <code>MediaSetClip</code> function whenever it changes your media's clipping region (see page 10-34 for more information about this function). The <code>trackMatte</code> field in this structure specifies a matte region.</p>
<code>height</code>	Indicates the height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.
<code>trackMovieMatrix</code>	<p>Specifies the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media.</p> <p>Whenever the Movie Toolbox modifies this matrix, it calls your derived media handler's <code>MediaSetMatrix</code> function, which is discussed on page 10-33.</p>
<code>moviePort</code>	Indicates the movie's graphics port. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's <code>MediaSetGWorld</code> function, which is discussed on page 10-31.
<code>movieGD</code>	Specifies the movie's graphics device. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's <code>MediaSetGWorld</code> function, which is discussed on page 10-31.
<code>trackMatte</code>	Identifies the matte region assigned to the track that uses your media. This field contains a handle to a pixel map that contains a blend matte. Your component is not responsible for disposing of this matte. If there is no matte, this field is set to <code>nil</code> .

Functions

This section describes the functions that may be supported by derived media handler components. It is divided into the following topics:

- “Managing Your Media Handler Component” discusses the functions that allow the Movie Toolbox to manage its connection to your component.
- “General Data Management” describes the functions that allow the Movie Toolbox to manage the general characteristics of the control path through your component.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

- “Graphics Data Management” tells you about the functions that allow the Movie Toolbox to manage the graphical characteristics of the control path through your component.
- “Sound Data Management” provides information about the function that allows the Movie Toolbox to manage the sound characteristics of the control path through your component.
- “Base Media Handler Utility Function” discusses a function that allows your derived media handler to report its capabilities to the base media handler.

Note

Many of the functions described in this chapter are optional—that is, your derived media handler may not need to support them. The description of each function discusses the issues you should consider when deciding whether or not to support a specific function. ♦

Managing Your Media Handler Component

Derived media handlers provide three functions that allow the Movie Toolbox to manage its relationship with the media handler. The Movie Toolbox calls your `MediaInitialize` function in order to give you an opportunity to prepare to provide access to your media. The Movie Toolbox grants processing time to your handler by calling your `MediaIdle` function. Your `MediaGetStatus` function allows the Movie Toolbox to retrieve status information after calling `MediaIdle`.

MediaInitialize

The `MediaInitialize` function allows your derived media handler component to prepare itself for providing access to its media.

```
pascal ComponentResult MediaInitialize (ComponentInstance ci,
                                       GetMovieCompleteParams *gmc);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>gmc</code>	Contains a pointer to a complete movie parameter structure, which is described in “Data Type” beginning on page 10-15. You can obtain information about the current media from this structure. You should copy any values you need to save into your derived media handler's local data area. Because this data structure is owned by the Movie Toolbox, you do not need to worry about disposing of any of the data in it.

DESCRIPTION

Once the Movie Toolbox has loaded a movie's data from its file, the toolbox calls your derived media handler's `MediaInitialize` function. If the user is creating a new

10-18 `pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /* remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived Media Handler Components Reference`

movie, the Movie Toolbox calls your media handler anyway, even though there may be no media data.

This function gives your media handler an opportunity to get ready to support the Movie Toolbox. As part of these preparations, your derived media handler should report its capabilities to the base media handler by calling the `MediaSetHandlerCapabilities` function (see page 10-38 for more information about this function).

You may choose to examine the data in the movie parameter structure; you may also save values from this structure. If you save references to structures (such as the matte pixel map), do not dispose of the memory associated with these structures. The Movie Toolbox owns these structures.

All derived media handlers should support this function. In addition, if your media handler saves values from the movie parameter structure that may change, be sure to support the corresponding functions that allow the Movie Toolbox to report changes to your media handler. For example, if your handler saves the movie time scale from the `movieScale` field, you should also support the `MediaSetMovieTimeScale` function.

If you return an error, the Movie Toolbox disables the track that uses your media. In cases where your media has just been created, the Movie Toolbox immediately disposes of your media.

Note that the Movie Toolbox may call other functions supported by your media handler before it calls your `MediaInitialize` function. In particular, it may call your `MediaGetMediaInfo` and `MediaPutMediaInfo` functions. However, before the Movie Toolbox tries to do anything with the data in your media, it will call your `MediaInitialize` function. The Movie Toolbox loads the movie's data using functions that are supported by the base media handler—your media handler does not have to support those functions.

RESULT CODES

Any Component Manager result code

MediaIdle

The `MediaIdle` function allows the Movie Toolbox to provide processing time to your derived media handler during movie playback. Your media handler may use this time to play its media.

```
pascal ComponentResult MediaIdle (ComponentInstance ci,
                                  TimeValue atMediaTime,
                                  long flagsIn, long *flagsOut,
                                  const TimeRecord *movieTime);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

`atMediaTime`

Specifies the current time, in your media's time base. You can use this time to determine the appropriate media data to work with.

`flagsIn`

Contains flags that indicate what the Movie Toolbox wants your media handler to do. These flags are applicable only to media handlers that perform their own scheduling.

The following flags are defined—the Movie Toolbox may use none, or it may set one or more flag to 1:

`mMustDraw`

Indicates that your media handler must play its media at this time. For graphical media, this means that your handler must draw the appropriate media data on the screen. For sound-based media, your handler must play the media's sounds. If this flag is set to 1, the Movie Toolbox has encountered a new sample in your media.

`mAtEnd`

Indicates that the current time corresponds to the end of the movie.

`mPreflightDraw`

Indicates that your media handler must not play its media at this time. Your handler may examine the media data and prepare to play it, but you should not draw any graphical data or play any sounds. If this flag is set to 1, your handler must not play its data.

If these flags are set to 0, then your media handler is free to decide whether to play the media data or not.

`flagsOut`

Contains a pointer to a long integer that your media handler uses to indicate to the Movie Toolbox what the handler did. You must always set the values of these flags appropriately.

The following flags are defined:

`mDidDraw`

Indicates that your media handler played its media's data with the `handlerHasSpatial` flag set, then it drew the data. Any time your media handler plays its media's data, you should set this flag to 1 when you return from your `MediaIdle` function. The Movie Toolbox uses this information when it is displaying a composited movie—that is, a movie whose image is derived by blending several tracks together. If your media's track is obscured by other, semitransparent tracks, the Movie Toolbox must redraw those other tracks whenever your media's image changes.

`mNeedsToDraw`

Indicates that your media handler needs to play its data. Typically, you use this flag when the Movie Toolbox calls your `MediaIdle` function with the `mPreflightDraw` flag in the `flagsIn` field set to 1, and you discover that you have data that must be played at the current time. Set this flag to 1 if your handler needs to play its media's data.

Derived Media Handler Components

`movieTime` Contains a pointer to the movie time value corresponding to the `atMediaTime` parameter. Note that this may differ from the current value returned by the Movie Toolbox's `GetMovieTime` function.

DESCRIPTION

The Movie Toolbox uses your `MediaIdle` function to provide processing time to your derived media handler during movie playback. Your media handler is free to use this time in any appropriate manner. For example, if your media handler supports a sound data type, you might prepare to play your media's sounds or actually play them, depending upon the options asserted by the Movie Toolbox. Your media handler is responsible for limiting the amount of processing time it uses.

The Movie Toolbox provides the current time, in your media's time base, in the `atMediaTime` parameter. You can use this value to obtain the appropriate samples and sample descriptions from your media (using the Movie Toolbox's `GetMediaSample` function). Your media handler may then work with the sample data and descriptions as appropriate.

If you encounter an error, save the result code. The Movie Toolbox polls you for status information using the `MediaGGetStatus` function, which is described next.

Your handler should examine the `flagsIn` parameter each time the Movie Toolbox calls its `MediaIdle` function. The flags in this parameter indicate the actions that your handler may perform. In addition, when you return from your `MediaIdle` function, you should report what you did using the `flagsOut` parameter. You tell the base media handler that you perform your own scheduling by setting the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function (see page 10-38 for more information about this function).

If your media handler changes any of the settings of the movie's graphics port or graphics world, be sure to restore the original settings before you exit. In addition, note that you may be drawing into a black-and-white graphics port. Finally, be aware that the Movie Toolbox also uses this function to obtain data for QuickDraw pictures. Therefore, if your media handler does not use QuickDraw when drawing to the screen, be sure to examine the `picSave` field in the graphics port so that you can detect when the toolbox wants to save an image. Your media handler is then responsible for performing the appropriate display processing. (For details on QuickDraw pictures, see the chapter "Basic QuickDraw" in *Inside Macintosh: Imaging*.)

Your derived media handler should support this function if you need to do work during movie playback. If you set the `handlerNoIdle` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function, the Movie Toolbox does not call your `MediaIdle` function.

RESULT CODES

Any Component Manager result code

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

MediaGGetStatus

The `MediaGGetStatus` function allows your derived media handler to report error conditions to the Movie Toolbox.

```
pascal ComponentResult MediaGGetStatus (ComponentInstance ci,
                                         ComponentResult *statusErr);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>statusErr</code>	Contains a pointer to a component result field. If you have error information that you would like to report to the Movie Toolbox, place an appropriate result code into the field referred to by this pointer.

DESCRIPTION

The Movie Toolbox calls your `MediaGGetStatus` function whenever an application calls the toolbox's `GetMovieStatus` or `GetTrackStatus` function. This provides your media handler an opportunity to report any difficulties it may be having in playing your media. You should use this mechanism to report any errors you encounter in your `MediaIdle` function (described in the previous section). You may use any appropriate result code.

Your derived media handler should support this function if you anticipate that you may encounter an error when playing your media. Because these errors may include such conditions as low memory or missing hardware, you should only rarely create a derived media handler that does not support this function. If your media handler does not support this function, the base media handler always sets the returned result code to `noErr`.

RESULT CODES

Any Component Manager result code

General Data Management

While the base media handler isolates your component from the details of media data access, your derived media handler still needs to keep track of certain information in order to support movie playback and creation. This section discusses functions that help your media handler manage its information.

Your media handler may store proprietary information in its media. The Movie Toolbox calls two media handler functions in order to give you an opportunity to retrieve or store this information. The `MediaPutMediaInfo` function allows you to store your special information in your media. The `MediaGetMediaInfo` function delivers that data to your media handler.

The Movie Toolbox tells your media handler when its track has been enabled or disabled by calling your `MediaSetActive` function. The Movie Toolbox prepares your handler

for playback by calling your `MediaPreroll` function. Whenever your media's playback rate changes, the Movie Toolbox calls your `MediaSetRate` function. Whenever the track that uses your media is edited, the Movie Toolbox calls your `MediaTrackEdited` function.

If the Movie Toolbox has called its `SetMediaSampleDescription` function on a sample description, it uses the `MediaSampleDescriptionChanged` function to notify your media handler of the change.

The Movie Toolbox allows tracks to be identified by various characteristics. For instance, it is possible to request that all tracks containing audio information be searched. To determine whether a track has a given characteristic, the Movie Toolbox queries the media handler for each track. The Movie Toolbox calls the `MediaHasCharacteristic` function with the specified characteristic.

The Movie Toolbox uses two functions to inform you about changes to your media's time environment. The `MediaSetMediaTimeScale` function allows the Movie Toolbox to change your media's time scale. The `MediaSetMovieTimeScale` function allows the Movie Toolbox to tell you when the movie's time scale has changed.

MediaPutMediaInfo

The `MediaPutMediaInfo` function allows your derived media handler to store proprietary information in its media.

```
pascal ComponentResult MediaPutMediaInfo (ComponentInstance ci,
                                           Handle h);
```

ci	Identifies the Movie Toolbox's connection to your derived media handler.
h	Contains a handle to storage into which your media handler may place its proprietary information. You determine the format and content of the data that you store in this handle. Your media handler must resize the handle as appropriate before you exit this function. Do not dispose of this handle—it is owned by the Movie Toolbox. The Movie Toolbox uses the base media handler to write this data to your media.

DESCRIPTION

The Movie Toolbox uses the `MediaPutMediaInfo` function to provide you an opportunity to store private data in your media. You determine the format and content of this data. The base media handler stores some information for you, including the media's transfer mode, opcolor, and sound balance. However, you may need to store additional information. For example, you may want to place a version number in each media you create.

Whenever the Movie Toolbox opens your media, it provides this private data to your media handler by calling your `MediaGetMediaInfo` function, which is described next.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

Note that the Movie Toolbox may call this function before it calls your `MediaInitialize` function.

Your derived media handler should support this function if you need to store private data in your media.

RESULT CODES

Any Component Manager result code

MediaGetMediaInfo

The `MediaGetMediaInfo` function allows your derived media handler to obtain the private data you have stored in your media.

```
pascal ComponentResult MediaGetMediaInfo (ComponentInstance ci,
                                           Handle h);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>h</code>	Contains a handle to storage containing your media handler's proprietary information. Your media handler creates this private data when the Movie Toolbox calls your <code>MediaPutMediaInfo</code> function. Do not dispose of this handle—it is owned by the Movie Toolbox.

DESCRIPTION

If you placed private data into your media, the Movie Toolbox calls your media handler's `MediaPutMediaInfo` function whenever it opens your media. Your media handler determines the format and content of this private data. Note that the Movie Toolbox may call this function before it calls your `MediaInitialize` function.

Your derived media handler should support this function if you store private data in your media.

RESULT CODES

Any Component Manager result code

MediaSetActive

The `MediaSetActive` function allows the Movie Toolbox to enable and disable your media.

```
pascal ComponentResult MediaSetActive (ComponentInstance ci,
                                       Boolean enableMedia);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

`enableMedia` Contains a Boolean value that indicates whether your media is enabled or disabled. If this parameter is set to `true`, your media is enabled; if the parameter is `false`, your media is disabled.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetActive` function whenever your media is either enabled or disabled. Initially, your media is disabled. Subsequently, the enabled state of your media is controlled by the state of the track that is using your media. When that track is enabled, your media is enabled; when that track is disabled, your media is disabled. Applications can control the enabled state of a track by using the Movie Toolbox's `SetTrackEnabled` function.

Your derived media handler should support this function if you perform your own scheduling or if your media handler uses significant amounts of temporary storage. If you are doing your own scheduling (that is, you have set the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function—see page 10-38 for more information about this function), your media handler needs to keep account of the media's active state so that you can properly respond to Movie Toolbox requests. When your media is disabled, you may choose to dispose of temporary storage you have allocated, so that the storage is available to other programs.

RESULT CODES

Any Component Manager result code

MediaPreroll

The `MediaPreroll` function allows the Movie Toolbox to prepare your media handler for playback.

```
pascal ComponentResult MediaPreroll (ComponentInstance ci,
                                       TimeValue time, Fixed rate);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

<code>time</code>	Contains the starting time of the media segment to play. This time value is expressed in your media's time scale.
<code>rate</code>	Specifies the rate at which the Movie Toolbox expects to play the media. This is a 32-bit, fixed-point number. Positive values indicate forward rates; negative values correspond to reverse rates.

DESCRIPTION

Use this as an opportunity to load data from your media, so that when the Movie Toolbox starts to play, your media can play smoothly from the start.

RESULT CODES

Any Component Manager result code

MediaSetRate

The `MediaSetRate` function allows the Movie Toolbox to set your media's playback rate.

```
pascal ComponentResult MediaSetRate (ComponentInstance ci,
                                     Fixed rate);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>rate</code>	Contains a 32-bit, fixed-point number that indicates your media's new effective playback rate. This effective rate accounts for any master time bases that may be in use with the current movie. Positive values represent forward rates and negative values indicate reverse rates.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetRate` function whenever the movie's playback rate changes. The Movie Toolbox provides you with a new effective rate for your media. This effective rate accounts for any master time bases that may be in use with the current movie. Consequently, you may use this rate without having to further transform it.

You obtain the initial rate information from the `effectiveRate` field of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function.

Your derived media handler should support this function if you perform your own scheduling. If you are doing your own scheduling (that is, you have set the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function—see page 10-38 for more information about this function), your media handler can use this function to determine when your

media is playing, and the direction and rate of playback. This information can help you prepare for playback more efficiently.

RESULT CODES

Any Component Manager result code

MediaTrackEdited

The `MediaTrackEdited` function allows the Movie Toolbox to inform your derived media handler about edits to its track.

```
pascal ComponentResult MediaTrackEdited (ComponentInstance ci);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaTrackEdited` function whenever the track that is using your media is edited. While these edits do not directly affect the data in your media, they can change the way in which the movie uses your media's data.

Your derived media handler should support this function if you are caching location information about track edits, or if you are using any time values in the movie's time base. Whenever the Movie Toolbox calls this function, your media handler should recalculate this type of information.

RESULT CODES

Any Component Manager result code

MediaSampleDescriptionChanged

The `MediaSampleDescriptionChanged` function allows the Movie Toolbox to inform your media handler that its `SetMediaSampleDescription` function has been called for a specified sample description.

```
pascal ComponentResult MediaSampleDescriptionChanged
                                (ComponentInstance ci,
                                long index);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

`index` Specifies the index of the sample description that has been changed.

DESCRIPTION

If your media handler caches sample description structures for any reason, it should support the `MediaSampleDescriptionChanged` function so that it will know when to update or invalidate the contents of that cache.

RESULT CODES

Any Component Manager result code

MediaHasCharacteristic

The Movie Toolbox calls the `MediaHasCharacteristic` function with a specified characteristic to allow tracks to be identified by various attributes.

```
pascal ComponentResult MediaHasCharacteristic
                                (ComponentInstance ci,
                                OSType characteristic,
                                Boolean *hasIt);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

`characteristic` Contains a constant that specifies the attribute of a track. Examples of characteristics that are currently defined are the Movie Toolbox constants `VisualMediaCharacteristic` and `AudioMediaCharacteristic`.

`hasIt` Contains a pointer to a Boolean value that specifies whether the track has the attribute specified in the `characteristic` parameter. Set this value to `true` if the attribute applies to your media handler; otherwise, set this value to `false`.

DESCRIPTION

The Movie Toolbox might request the search of all tracks with audio data. For example, to find out if a track has a given attribute, the Movie Toolbox queries the media handler for each track by calling `MediaHasCharacteristic` with a particular constant specified in the `characteristic` parameter. If your media handler does not recognize a characteristic, return a value of `false`.

You should implement this function for any media handler that has characteristics in addition to spatial ones. If you have set the `handlerHasSpatial` capabilities flag, the base media handler automatically handles the `VisualMediaCharacteristic` constant for you.

RESULT CODES

Any Component Manager result code

MediaSetMediaTimeScale

The `MediaSetMediaTimeScale` function allows the Movie Toolbox to inform your media handler that your media's time scale has been changed.

```
pascal ComponentResult MediaSetMediaTimeScale
                                (ComponentInstance ci,
                                 TimeScale newTimeScale);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

`newTimeScale` Specifies your media's new time scale.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetMediaTimeScale` function whenever your media's time scale is changed. Applications can change your media's time scale by using the Movie Toolbox's `SetMediaTimeScale` function. When the Movie Toolbox calls this function, your media handler should recalculate any time values you have stored that are expressed in your media's time coordinate system. Changing your media's time scale may also affect media playback.

You obtain the initial media time scale information from the `mediaScale` field of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function.

Your derived media handler should support this function if your media handler stores time information that pertains to its media.

RESULT CODES

Any Component Manager result code

MediaSetMovieTimeScale

The `MediaSetMovieTimeScale` function allows the Movie Toolbox to inform your media handler that the movie's time scale has been changed.

```
pascal ComponentResult MediaSetMovieTimeScale
                                (ComponentInstance ci,
                                 TimeScale newTimeScale);
```

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

`newTimeScale` Specifies the movie's new time scale.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetMovieTimeScale` function whenever the movie's time scale is changed. Applications can change the movie's time scale by using the Movie Toolbox's `SetMovieTimeScale` function. When the Movie Toolbox calls this function, your media handler should recalculate any time values you have stored that are expressed in the movie's time coordinate system. Changing the movie's time scale may also affect playback of your media.

You obtain the initial movie time scale information from the `movieScale` field of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function.

Your derived media handler should support this function if your media handler stores time information in the movie's time coordinate system.

RESULT CODES

Any Component Manager result code

Graphics Data Management

If your media handler draws media data on the screen, you need to manage your media's graphics environment. The Movie Toolbox uses a number of functions to inform you about changes to the graphics environment. The Movie Toolbox only calls these functions if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function.

The Movie Toolbox calls your handler's `MediaSetGWorld` function whenever your media's graphics port or graphics device has changed. The `MediaSetDimensions` function allows the Movie Toolbox to inform your handler about changes to its spatial dimensions. Whenever either the movie or track matrix changes, the Movie Toolbox calls your `MediaSetMatrix` function. Similarly, if your media's clipping region changes, the Movie Toolbox calls your `MediaSetClip` function.

When it is building up a movie's image from its component tracks, the Movie Toolbox must be able to determine which tracks are transparent. The Movie Toolbox calls your `MediaGetTrackOpaque` function to retrieve this information about your media.

The Movie Toolbox calls your `MediaGetNextBoundsChange` function so that it can learn when your media will next change its display shape. When the Movie Toolbox wants to find out the shape of the region into which you draw your media, it calls your `MediaGetSrcRgn` function.

MediaSetGWorld

The `MediaSetGWorld` function allows your derived media handler to learn about changes to your media's graphic environment.

```
pascal ComponentResult MediaSetGWorld (ComponentInstance ci,
                                       CGrafPtr aPort,
                                       GDHandle aGD);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>aPort</code>	Contains a pointer to the new graphics port. Note that this may be either a color or a black-and-white port.
<code>aGD</code>	Contains a handle to the new graphics device.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetGWorld` function whenever your media's graphics world changes. The toolbox provides you with the new graphics port and graphics device. You should then use this information for subsequent graphics operations.

Your derived media handler should support this function if you perform specialized graphics processing or if you are using the Image Compression Manager to decompress your media. Note that when the Movie Toolbox calls your `MediaIdle` function, it supplies you with information about the current graphics environment. Consequently, you do not need to support the `MediaSetGWorld` function in order to draw during playback. However, if your media data is compressed and you are using the Image Compression Manager to decompress sequences, you may need to provide updated graphics environment information before playback.

You obtain the initial graphics environment information from the `moviePort` and `movieGD` fields of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function.

RESULT CODES

Any Component Manager result code

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

MediaSetDimensions

The `MediaSetDimensions` function allows the Movie Toolbox to inform your media handler when its media's spatial dimensions change.

```
pascal ComponentResult MediaSetDimensions (ComponentInstance ci,
                                           Fixed width,
                                           Fixed height);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>width</code>	Indicates the width, in pixels, of the track rectangle. This field, along with the <code>height</code> field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.
<code>height</code>	Indicates the height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetDimensions` function whenever the spatial dimensions of your media's track change. The toolbox provides you with the dimensions of the rectangle that encloses your media's graphical image. Changes to this rectangle may affect the way in which you display your media's data.

You obtain the initial dimension information from the `width` and `height` fields of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function (described on page 10-18).

Your derived media handler should support this function if you draw during playback.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function (described on page 10-38).

RESULT CODES

Any Component Manager result code

SEE ALSO

The Movie Toolbox uses the `MediaSetMatrix` function (described in the next section) to tell your media handler about changes to either the movie matrix or the track matrix. In addition, your media handler's `MediaSetClip` function allows you to learn about changes to your media's clipping region. This function is discussed on page 10-34.

MediaSetMatrix

The `MediaSetMatrix` function allows the Movie Toolbox to tell your media handler about changes to either the movie matrix or the track matrix.

```
pascal ComponentResult MediaSetMatrix (ComponentInstance ci,
                                       const MatrixRecord *trackMovieMatrix);
```

`ci` Identifies the Movie Toolbox's connection to your derived media handler.

`trackMovieMatrix`

Contains a pointer to the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetMatrix` function whenever either the movie matrix or track matrix changes. The toolbox provides you with a matrix that concatenates the transformations defined by both the movie and track matrices. You can use this matrix to map your media's display representation into the movie's coordinate system. For example, by applying this matrix to the track rectangle, you can determine the display boundaries of your media (the track rectangle is defined by the `width` and `height` fields in the movie parameter structure that you obtain when the toolbox calls your `MediaInitialize` function).

You obtain the initial matrix from the `trackMovieMatrix` field of the movie parameter structure that the Movie Toolbox provides to your `MediaInitialize` function.

Your derived media handler should support this function if you draw during playback.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function (described on page 10-38).

SPECIAL CONSIDERATIONS

Before you try to use this matrix, you should make sure that your media handler can accommodate its transformations. You can use the Movie Toolbox's `GetMatrixType` function to learn about the matrix. If the matrix includes transformations that are beyond the capabilities of your media handler, you can direct the base media handler to do display processing on your behalf. Call the `MediaSetHandlerCapabilities` function and set the `handlerNeedsBuffer` flag to 1 in the `flags` parameter. This forces the base media handler to draw your media into an offscreen buffer.

RESULT CODES

Any Component Manager result code

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

SEE ALSO

The Movie Toolbox uses the `MediaSetDimensions` function to tell your media handler about changes to the rectangle that surrounds the graphical representation of your media; this function is described in the previous section. In addition, your media handler's `MediaSetClip` function allows you to learn about changes to your media's clipping region. This function is discussed next.

MediaSetClip

The `MediaSetClip` function allows your derived media handler to learn about changes to its clipping region.

```
pascal ComponentResult MediaSetClip (ComponentInstance ci,
                                     RgnHandle theClip);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>theClip</code>	Contains a handle to your media's clipping region. Your media handler is responsible for disposing of this region when you are done with it. Note that this region lies in the movie's coordinate system.

DESCRIPTION

The Movie Toolbox calls your derived media handler's `MediaSetClip` function whenever the track's clipping region changes. The toolbox provides you with a handle to a clipping region that supersedes any other clipping region you may be using.

Your derived media handler should support this function if you draw during playback.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` and `handlerCanClip` flags to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function (described on page 10-38).

RESULT CODES

Any Component Manager result code

MediaGetTrackOpaque

The `MediaGetTrackOpaque` function allows the Movie Toolbox to determine whether your media is transparent or opaque when displayed.

```
pascal ComponentResult MediaGetTrackOpaque (ComponentInstance ci,
                                             Boolean *trackIsOpaque);
```


Derived Media Handler Components

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>trackIsOpaque</code>	Contains a pointer to a Boolean value. Your media handler must set this Boolean value to indicate whether your media is transparent or opaque when displayed. Set the Boolean value to <code>true</code> if your media is semitransparent (that is, you draw in blend mode); otherwise, leave the flag unchanged.

DESCRIPTION

The Movie Toolbox uses this function when it is building a movie from composited images. Your media handler returns information that tells the toolbox whether your media's displayed image is to be combined with other images that are already on the screen. If you draw your media in blend mode, for example, your media is semitransparent, and its display relies upon other images on the screen. The Movie Toolbox needs to know this in order to correctly display the movie to the user.

Your derived media handler should support this function if your media is semitransparent when displayed or if you handle display transfer modes.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` or `handlerCanTransferMode` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function.

RESULT CODES

Any Component Manager result code

MediaGetNextBoundsChange

The `MediaGetNextBoundsChange` function allows the Movie Toolbox to determine when your media causes a spatial change to the movie.

```
pascal ComponentResult MediaGetNextBoundsChange
                                (ComponentInstance ci,
                                 TimeValue *when);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>when</code>	Contains a pointer to a movie time value. Your media handler must set this time value. Be sure to return a time value in the movie's time base. Use the current effective rate to determine the direction your media is playing. Set this value to -1 if there are no more changes in the specified direction.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

DESCRIPTION

The Movie Toolbox uses this function to determine when the next spatial change will occur in the current movie. Your media handler returns a time value. Your media handler must examine the edit list of the track that contains your media in order to derive this duration. You can use the Movie Toolbox's `GetTrackNextInterestingTime` function to retrieve time values in the movie's time coordinate system. For details on this function and on movie time values, see the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime*.

Your derived media handler should support this function if you change the shape of your media's spatial representation during playback.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function.

RESULT CODES

Any Component Manager result code

MediaGetSrcRgn

The `MediaGetSrcRgn` function allows your derived media handler to specify an irregular destination display region to the Movie Toolbox.

```
pascal ComponentResult MediaGetSrcRgn (ComponentInstance ci,
                                       RgnHandle rgn,
                                       TimeValue atMediaTime);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your derived media handler.
<code>rgn</code>	Contains a handle to a region. When the Movie Toolbox calls your function, this region is initialized to the track's boundary rectangle (which is defined by the <code>width</code> and <code>height</code> fields in the movie parameter structure that you obtain when the toolbox calls your <code>MediaInitialize</code> function, which is described on page 10-18). Your media handler may then alter this region as appropriate, so that it corresponds to the boundaries of your media's display image. Note that this region is in the track's coordinate system, not the movie's. Do not dispose of this region—it is owned by the Movie Toolbox.
<code>atMediaTime</code>	Specifies the time value at which the Movie Toolbox wants to know what the source region is.

DESCRIPTION

The Movie Toolbox uses this function to determine whether your media has an irregularly shaped display area. If your media is displayed in a nonrectangular area, or if

your media uses only a portion of the track rectangle, you can use this function to report that fact to the toolbox.

Your derived media handler should support this function if your media does not completely fill the track rectangle during playback.

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` function.

RESULT CODES

Any Component Manager result code

Sound Data Management

The Movie Toolbox uses your `MediaGSetVolume` function to tell your media handler when its sound volume has changed.

MediaGSetVolume

The `MediaGSetVolume` function allows your derived media handler to learn about changes to its sound volume setting.

```
pascal ComponentResult MediaGSetVolume (ComponentInstance ci,
                                         short volume);
```

ci	Identifies the Movie Toolbox's connection to your derived media handler.
volume	<p>Contains the media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.</p> <p>The Movie Toolbox scales your media's volume in light of the track's and movie's volume settings, but it does not take into account the system speaker volume setting. This value is appropriate for use with the Sound Manager.</p>

DESCRIPTION

The Movie Toolbox uses this function to tell your derived media handler about changes to your media's sound volume.

Your derived media handler should support this function if it can play sounds.

RESULT CODES

Any Component Manager result code

Base Media Handler Utility Function

Apple's base media handler component provides a utility function, `MediaSetHandlerCapabilities`, which allows you to tell the base handler what your derived handler can do.

MediaSetHandlerCapabilities

The `MediaSetHandlerCapabilities` function allows your derived media handler to report its capabilities to the base media handler.

```
pascal ComponentResult MediaSetHandlerCapabilities
                                (ComponentInstance ci,
                                long flags,
                                long flagsMask);
```

ci Identifies your derived media handler's connection to the base media handler.

flags Specifies the capabilities of your derived media handler. This parameter contains a number of flags, each of which corresponds to a particular feature. You may work with more than one flag at a time. The following flags are defined (be sure to set unused flags to 0):

`handlerHasSpatial`

Indicates that your handler does spatial processing. If you set this flag to 1, the Movie Toolbox informs your derived media handler about changes to the graphics environment or spatial representation of your media.

`handlerCanClip`

Indicates that your media handler can perform clipping. If you set this flag to 1, the Movie Toolbox calls your `MediaSetClip` function (described on page 10-34) whenever the clipping region changes.

`handlerCanMatte`

Reserved for Apple. Do not set this flag to 1.

`handlerCanTransferMode`

Indicates that you can work with transfer modes other than source copy or dither copy. If you set this flag to 1, the Movie Toolbox calls your `MediaGetTrackOpaque` function to determine whether your track is transparent.

Derived Media Handler Components

<code>handlerNeedsBuffer</code>	Indicates that your media handler needs help during playback. If you set this flag to 1, the base media handler allocates an offscreen buffer and handles all display transformations for you.
<code>handlerNoIdle</code>	Indicates that your derived media handler does not need any processing time during playback. If you set this flag to 1, the Movie Toolbox never calls your <code>MediaIdle</code> function. This is useful for media handlers that store data in a media, but do not play that data.
<code>handlerNoScheduler</code>	Indicates that your media handler performs special processing during playback. Normally, the Movie Toolbox calls your <code>MediaIdle</code> function only when it is time for your handler to draw data from a new media sample. If you set this flag to 1, the Movie Toolbox calls that function other times as well, so that your media handler can prepare for playback or perform other necessary processing.
<code>handlerWantsTime</code>	Indicates that your media handler needs additional processing time. If you set this flag to 1, the Movie Toolbox calls your <code>MediaIdle</code> function as often as possible.
<code>handlerCGrafPortOnly</code>	Indicates that your media handler can only draw into color graphics ports. If you set this flag to 1, the base media handler performs the necessary processing to display your color media on a black-and-white graphics port (this involves drawing to an offscreen buffer and then transferring the image to the screen).
<code>flagsMask</code>	Indicates which flags in the <code>flags</code> parameter are to be considered in this operation. For each bit in the <code>flags</code> parameter that you want the base media handler to consider, you must set the corresponding bit in the <code>flagsMask</code> parameter to 1. Set unused flags to 0. This allows you to work with a single flag without altering the settings of other flags.

DESCRIPTION

Use the `MediaSetHandlerCapabilities` function to tell the base media handler what your derived media handler can do. By default, all of the flags are set to 0—in this case, your media handler is only responsible for storing and retrieving data. You can specify further capabilities by setting various flags to 1. For example, if your handler draws data on the screen, be sure to set the `handlerHasSpatial` flag to 1. Other flags govern more detailed aspects of handler operation.

This function uses both a `flags` parameter and a `flagsMask` parameter. You specify which flags are to be changed in a given operation by setting the `flagsMask` parameter. You then specify the new values for those affected flags with the `flags` parameter.

```
pascal ComponentResult PictSetMatrix (PictGlobals store, MatrixRecord *trackMovieMatrix) { /*
remember the new display matrix */ store->matrix = *trackMovieMatrix; return noErr; }Derived
Media Handler Components Reference
```

Derived Media Handler Components

In this manner, you can work with a single flag without affecting the settings of any other flags.

Your media handler may call this function at any time. In general, you should call it from your `MediaInitialize` function (described on page 10-18), so that you report your capabilities to the base media handler before the Movie Toolbox starts working with your media. You may call this function again later, in response to changing conditions. For example, if your media handler receives a matrix that it cannot accommodate from the `MediaSetMatrix` function, you can allow the base media handler to handle your drawing by calling this function and setting the `handlerNeedsBuffer` flag in both the `flags` and `flagsMask` parameters to 1.

Note that this function is provided by the base media handler—your media handler does not support this function.

RESULT CODES

Any Component Manager result code

Summary of Derived Media Handler Components

C Summary

Constants

```

/* flags in flags parameter of MediaSetHandlerCapabilities function */
enum {
    handlerHasSpatial      = 1<<0,      /* draws */
    handlerCanClip         = 1<<1,      /* clips */
    handlerCanMatte        = 1<<2,      /* reserved */
    handlerCanTransferMode = 1<<3,      /* does transfer modes */
    handlerNeedsBuffer     = 1<<4,      /* use offscreen buffer */
    handlerNoIdle          = 1<<5,      /* never draws */
    handlerNoScheduler     = 1<<6,      /* schedules self */
    handlerWantsTime       = 1<<7,      /* needs more time */
    handlerCGrafPortOnly   = 1<<8      /* color only */
};

/* values for inFlags parameter of MediaIdle function */
enum {
    mMustDraw              = 1<<3,      /* must draw now */
    mAtEnd                 = 1<<4,      /* current time corresponds to
                                         end of movie */
    mPreflightDraw         = 1<<5      /* must not draw */
};

/* values for outFlags parameter of MediaIdle function */
enum {
    mDidDraw               = 1<<0,      /* did draw */
    mNeedsToDraw           = 1<<2      /* needs to draw */
};

/* component type and subtype values */
#define MediaHandlerType   'mhlr'      /* derived media handler */
#define BaseMediaHandlerType 'gnrc'    /* base media handler */

```

Derived Media Handler Components

```

/* constants used in the characteristic parameter of the
   MediaHasCharacteristic function */
#define VisualMediaCharacteristic 'eyes'    /* visual media characteristic */
#define AudioMediaCharacteristic 'ears'    /* audio media characteristic */

/* selectors for derived media handler components */
enum {
    enum {
        kMediaInitializeSelect              = 0x501, /* MediaInitialize */
        kMediaSetHandlerCapabilitiesSelect   = 0x502,
                                           /* MediaSetHandlerCapabilities */
        kMediaIdleSelect                    = 0x503, /* MediaIdle */
        kMediaGetMediaInfoSelect             = 0x504, /* MediaGetMediaInfo */
        kMediaPutMediaInfoSelect             = 0x505, /* MediaPutMediaInfo */
        kMediaSetActiveSelect               = 0x506, /* MediaSetActive */
        kMediaSetRateSelect                  = 0x507, /* MediaSetRate */
        kMediaGGetStatusSelect               = 0x508, /* MediaGGetStatus */
        kMediaTrackEditedSelect              = 0x509, /* MediaTrackEdited */
        kMediaSetMediaTimeScaleSelect        = 0x50A, /* MediaSetMediaTimeScale */
        kMediaSetMovieTimeScaleSelect        = 0x50B, /* MediaSetMovieTimeScale */
        kMediaSetGWorldSelect                = 0x50C, /* MediaSetGWorld */
        kMediaSetDimensionsSelect            = 0x50D, /* MediaSetDimensions */
        kMediaSetClipSelect                  = 0x50E, /* MediaSetClip */
        kMediaSetMatrixSelect                = 0x50F, /* MediaSetMatrix */
        kMediaGetTrackOpaqueSelect           = 0x510, /* MediaGetTrackOpaque */
        kMediaSetGraphicsModeSelect          = 0x511, /* MediaSetGraphicsMode */
        kMediaGetGraphicsModeSelect          = 0x512, /* MediaGetGraphicsMode */
        kMediaGSetVolumeSelect               = 0x513, /* MediaGSetVolume */
        kMediaSetSoundBalanceSelect          = 0x514, /* MediaSetSoundBalance */
        kMediaGetSoundBalanceSelect          = 0x515, /* MediaGetSoundBalance */
        kMediaGetNextBoundsChangeSelect      = 0x516,
                                           /* MediaGetNextBoundsChange */
        kMediaGetSrcRgnSelect                = 0x517, /* MediaGetSrcRgn */
        kMediaPrerollSelect                  = 0x518, /* MediaPreroll */
        kMediaSampleDescriptionChangedSelect = 0x519,
                                           /* MediaSampleDescriptionChanged */
        kMediaHasCharacteristicSelect        = 0x51A /* MediaHasCharacteristic */
    };
};

```


Data Type

```
typedef struct {
    short      version;           /* version--always 0 */
    Movie      theMovie;          /* movie identifier */
    Track      theTrack;          /* track identifier */
    Media      theMedia;          /* media identifier */
    TimeScale  movieScale;        /* movie's time scale */
    TimeScale  mediaScale;        /* media's time scale */
    TimeValue  movieDuration;     /* movie's duration */
    TimeValue  trackDuration;     /* track's duration */
    TimeValue  mediaDuration;     /* media's duration */
    Fixed      effectiveRate;     /* media's effective rate */
    TimeBase   timeBase;          /* media's time base */
    short      volume;            /* media's volume */
    Fixed      width;             /* width of display area */
    Fixed      height;            /* height of display area */
    MatrixRecord trackMovieMatrix; /* transformation matrix */
    CGrafPtr   moviePort;         /* movie's graphics port */
    GDHandle   movieGD;           /* movie's graphics device */
    PixMapHandle trackMatte;      /* track's matte */
} GetMovieCompleteParams;
```

Functions

Managing Your Media Handler Component

```
pascal ComponentResult MediaInitialize
    (ComponentInstance ci,
     GetMovieCompleteParams *gmc);

pascal ComponentResult MediaIdle
    (ComponentInstance ci,
     TimeValue atMediaTime, long flagsIn,
     long *flagsOut, const TimeRecord *movieTime);

pascal ComponentResult MediaGGetStatus
    (ComponentInstance ci,
     ComponentResult *statusErr);
```

General Data Management

```
pascal ComponentResult MediaPutMediaInfo
    (ComponentInstance ci, Handle h);

pascal ComponentResult MediaGetMediaInfo
    (ComponentInstance ci, Handle h);
```

Derived Media Handler Components

```

pascal ComponentResult MediaSetActive
    (ComponentInstance ci, Boolean enableMedia);
pascal ComponentResult MediaPreroll
    (ComponentInstance ci, TimeValue time,
     Fixed rate);
pascal ComponentResult MediaSetRate
    (ComponentInstance ci, Fixed rate);
pascal ComponentResult MediaTrackEdited
    (ComponentInstance ci);
pascal ComponentResult MediaSampleDescriptionChanged
    (ComponentInstance ci, long index);
pascal ComponentResult MediaHasCharacteristic
    (ComponentInstance ci,
     OSType characteristic, Boolean *hasIt);
pascal ComponentResult MediaSetMediaTimeScale
    (ComponentInstance ci, TimeScale newTimeScale);
pascal ComponentResult MediaSetMovieTimeScale
    (ComponentInstance ci, TimeScale newTimeScale);

```

Graphics Data Management

```

pascal ComponentResult MediaSetGWorld
    (ComponentInstance ci, CGrafPtr aPort,
     GDHandle aGD);
pascal ComponentResult MediaSetDimensions
    (ComponentInstance ci, Fixed width,
     Fixed height);
pascal ComponentResult MediaSetMatrix
    (ComponentInstance ci,
     const MatrixRecord *trackMovieMatrix);
pascal ComponentResult MediaSetClip
    (ComponentInstance ci, RgnHandle theClip);
pascal ComponentResult MediaGetTrackOpaque
    (ComponentInstance ci, Boolean *trackIsOpaque);
pascal ComponentResult MediaGetNextBoundsChange
    (ComponentInstance ci, TimeValue *when);
pascal ComponentResult MediaGetSrcRgn
    (ComponentInstance ci, RgnHandle rgn,
     TimeValue atMediaTime);

```

Sound Data Management

```

pascal ComponentResult MediaGSetVolume
    (ComponentInstance ci, short volume);

```

Base Media Handler Utility Function

```
pascal ComponentResult MediaSetHandlerCapabilities
    (ComponentInstance ci, long flags,
     long flagsMask);
```

Pascal Summary

Constants

```
CONST
{flags in flags parameter of MediaSetHandlerCapabilities function}
    handlerHasSpatial      =  $1;          {draws}
    handlerCanClip         =  $2;          {clips}
    handlerCanMatte        =  $4;          {reserved}
    handlerCanTransferMode =  $8;          {does transfer modes}
    handlerNeedsBuffer     = $10;          {use offscreen buffer}
    handlerNoIdle          = $20;          {never draws}
    handlerNoScheduler     = $40;          {schedules self}
    handlerWantsTime       = $80;          {needs more time}
    handlerCGrafPortOnly   = $100;         {color only}

{values for inFlags parameter of MediaIdle function}
    mMustDraw              =  $8;          {must draw now}
    mAtEnd                  =  $10;         {current time corresponds to }
                                     { end of movie}
    mPreflightDraw         =  $20;         {must not draw}

{values for outFlags parameter of MediaIdle function}
    mDidDraw                =  $1;          {did draw}
    mNeedsToDraw            =  $4;          {needs to draw}

{component type and subtype values}
    MediaHandlerType        'mhlr'         {derived media handler}
    BaseMediaType           'gnrc'         {base media handler}

{constants used in the characteristic parameter of the }
{ MediaHasCharacteristic function}
    VisualMediaCharacteristic  'eyes'      {visual media characteristic}
    AudioMediaCharacteristic   'ears'      {audio media characteristic}
```

Derived Media Handler Components

```

{selectors for derived media handler components}
    kMediaInitializeSelect          = $501;    {MediaInitialize}
    kMediaSetHandlerCapabilitiesSelect = $502;    {MediaSetHandlerCapabilities}
    kMediaIdleSelect                = $503;    {MediaIdle}
    kMediaGetMediaInfoSelect        = $504;    {MediaGetMediaInfo}
    kMediaPutMediaInfoSelect        = $505;    {MediaPutMediaInfo}
    kMediaSetActiveSelect           = $506;    {MediaSetActive}
    kMediaSetRateSelect             = $507;    {MediaSetRate}
    kMediaGGetStatusSelect          = $508;    {MediaGGetStatus}
    kMediaTrackEditedSelect         = $509;    {MediaTrackEdited}
    kMediaSetMediaTimeScaleSelect   = $50A;    {MediaSetMediaTimeScale}
    kMediaSetMovieTimeScaleSelect   = $50B;    {MediaSetMovieTimeScale}
    kMediaSetGWorldSelect           = $50C;    {MediaSetGWorld}
    kMediaSetDimensionsSelect       = $50D;    {MediaSetDimensions}
    kMediaSetClipSelect             = $50E;    {MediaSetClip}
    kMediaSetMatrixSelect           = $50F;    {MediaSetMatrix}
    kMediaGetTrackOpaqueSelect       = $510;    {MediaGetTrackOpaque}
    kMediaSetGraphicsModeSelect     = $511;    {MediaSetGraphicsMode}
    kMediaGetGraphicsModeSelect     = $512;    {MediaGetGraphicsMode}
    kMediaGSetVolumeSelect          = $513;    {MediaGSetVolume}
    kMediaSetSoundBalanceSelect      = $514;    {MediaSetSoundBalance}
    kMediaGetSoundBalanceSelect      = $515;    {MediaGetSoundBalance}
    kMediaGetNextBoundsChangeSelect = $516;    {MediaGetNextBoundsChange}
    kMediaGetSrcRgnSelect           = $517;    {MediaGetSrcRgn}
    kMediaPrerollSelect             = $518;    {MediaPreroll}
    kMediaSampleDescriptionChangedSelect = $519;    {MediaSampleDescriptionChanged}
    kMediaHasCharacteristicSelect    = $51A;    {MediaHasCharacteristic}

```

Data Type

TYPE

```
GetMovieCompleteParams =
```

RECORD

version:	Integer;	{version; always 0}
theMovie:	Movie;	{movie identifier}
theTrack:	Track;	{track identifier}
theMedia:	Media;	{media identifier}
movieScale:	TimeScale;	{movie's time scale}
mediaScale:	TimeScale;	{media's time scale}
movieDuration:	TimeValue;	{movie's duration}
trackDuration:	TimeValue;	{track's duration}
mediaDuration:	TimeValue;	{media's duration}

Derived Media Handler Components

```

effectiveRate:    Fixed;           {media's effective rate}
timeBase:         TimeBase;        {media's time base}
volume:           Integer;          {media's volume}
width:            Fixed;            {width of display area}
height:           Fixed;            {height of display area}
trackMovieMatrix: MatrixRecord;     {transformation matrix}
moviePort:        CGrafPtr;         {movie's graphics port}
movieGD:          GDHandle;         {movie's graphics device}
trackMatte:       PixMapHandle;     {track's matte}
END;

```

Routines

Managing Your Media Handler Component

```

FUNCTION MediaInitialize (ci: ComponentInstance;
                        VAR gmc: GetMovieCompleteParams):
                        ComponentResult;

FUNCTION MediaIdle (ci: ComponentInstance; atMediaTime: TimeValue;
                  flagsIn: LongInt; VAR flagsOut: LongInt;
                  VAR movieTime: TimeRecord): ComponentResult;

FUNCTION MediaGGetStatus (ci: ComponentInstance;
                        VAR statusErr: ComponentResult):
                        ComponentResult;

```

General Data Management

```

FUNCTION MediaPutMediaInfo (ci: ComponentInstance; h: Handle):
                        ComponentResult;

FUNCTION MediaGetMediaInfo (ci: ComponentInstance; h: Handle):
                        ComponentResult;

FUNCTION MediaSetActive (ci: ComponentInstance; enableMedia: Boolean):
                        ComponentResult;

FUNCTION MediaPreroll (ci: ComponentInstance; time: TimeValue;
                    rate: Fixed): ComponentResult;

FUNCTION MediaSetRate (ci: ComponentInstance; rate: Fixed):
                        ComponentResult;

FUNCTION MediaTrackEdited (ci: ComponentInstance): ComponentResult;

FUNCTION MediaSampleDescriptionChanged
                        (ci: ComponentInstance; index: LongInt):
                        ComponentResult;

```

Derived Media Handler Components

```

FUNCTION MediaHasCharacteristic
    (ci: ComponentInstance; characteristic: OSType;
     VAR hasIt: Boolean): ComponentResult;

FUNCTION MediaSetMediaTimeScale
    (ci: ComponentInstance;
     newTimeScale: TimeScale): ComponentResult;

FUNCTION MediaSetMovieTimeScale
    (ci: ComponentInstance;
     newTimeScale: TimeScale): ComponentResult;

```

Graphics Data Management

```

FUNCTION MediaSetGWorld    (ci: ComponentInstance; aPort: CGrafPtr;
                           aGD: GDHandle): ComponentResult;

FUNCTION MediaSetDimensions (ci: ComponentInstance; width: Fixed;
                             height: Fixed): ComponentResult;

FUNCTION MediaSetMatrix    (ci: ComponentInstance;
                           VAR trackMovieMatrix: MatrixRecord):
    ComponentResult;

FUNCTION MediaSetClip      (ci: ComponentInstance; theClip: RgnHandle):
    ComponentResult;

FUNCTION MediaGetTrackOpaque
    (ci: ComponentInstance;
     VAR trackIsOpaque: Boolean): ComponentResult;

FUNCTION MediaGetNextBoundsChange
    (ci: ComponentInstance; VAR when: TimeValue):
    ComponentResult;

FUNCTION MediaGetSrcRgn    (ci: ComponentInstance; rgn: RgnHandle;
                           atMediaTime: TimeValue): ComponentResult;

```

Sound Data Management

```

FUNCTION MediaGSetVolume    (ci: ComponentInstance; volume: Integer):
    ComponentResult;

```

Base Media Handler Utility Routine

```

FUNCTION MediaSetHandlerCapabilities
    (ci: ComponentInstance; flags: LongInt;
     flagsMask: LongInt): ComponentResult;

```