

This chapter discusses movie data exchange components. **Movie data exchange components** allow applications to move various types of data into and out of a QuickTime movie. These components provide data conversion services to and from standard QuickTime movie data formats. **Movie data import components** convert other data formats into QuickTime's movie data format; **movie data export components** convert QuickTime movie data into other formats.

This chapter is divided into the following sections:

- “About Movie Data Exchange Components” provides a general introduction to components of this type.
- “Using Movie Data Exchange Components” discusses how applications use these components.
- “Creating a Movie Data Exchange Component” describes how to create movie import and export components with sample programs for their implementation.
- “Movie Data Exchange Components Reference” presents detailed information about the functions that are supported by these components.
- “Summary of Movie Data Exchange Components” contains a condensed listing of the constants, data structures, and functions supported by these components.

This chapter addresses developers of movie data exchange components. If you plan to create either a movie data import component or a movie data export component (or both), you should read the entire chapter. If you are writing an application that uses components of this type, you should read the first two sections (“About Movie Data Exchange Components” and “Using Movie Data Exchange Components”), and consult “Movie Data Exchange Components Reference” as appropriate.

As components, movie data exchange components rely on the facilities of the Component Manager. In order 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. See “Movie Toolbox” in *Inside Macintosh: QuickTime* for more information.

About Movie Data Exchange Components

This section provides background information about movie data exchange components. After reading this section, you should understand why these components exist and whether you need to create or use one.

Movie data exchange components allow applications to place various types of data into a QuickTime movie or extract data from a movie in a specified format. Movie data import components translate foreign (that is, nonmovie) data formats into QuickTime movie data format. For example, a movie data import component might convert images from a paint application into frames in a QuickTime movie.

Movie Data Exchange Components

Conversely, movie data export components convert movie data into other formats, so that the data can be used by other applications. As an example, a movie data export component might allow an application to extract the sound track from a QuickTime movie in AIFF format. The extracted sound track may then be manipulated by applications that are not QuickTime-aware.

Applications use the services of movie data exchange components by calling the Movie Toolbox. Figure 9-1 shows the relationship between the Movie Toolbox and movie data import components; Figure 9-2 shows how movie data export components fit into the picture.

Figure 9-1 The Movie Toolbox, movie data import components, and your application

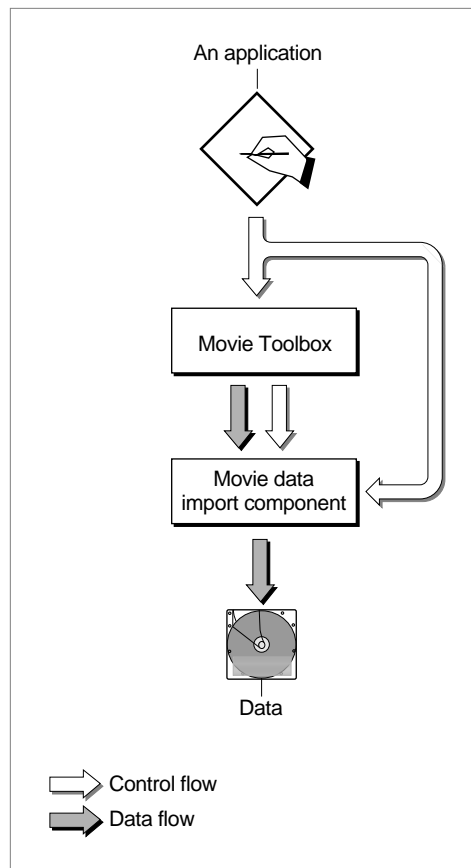
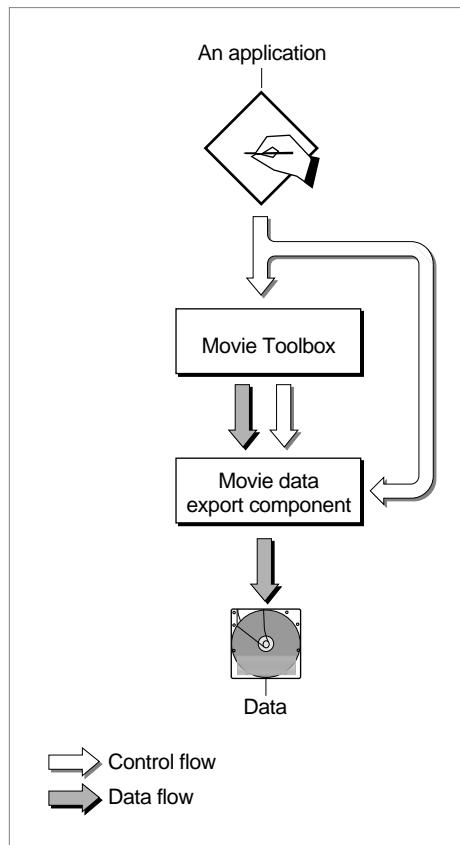


Figure 9-2 The Movie Toolbox, movie data export components, and your application

The next section describes in detail how to use each of these components.

If you are writing a media handler that works with a new type of data, you will probably need to use one or more data exchange components to facilitate the importing and exporting of data to QuickTime movies.

Using Movie Data Exchange Components

This section discusses how applications use movie data exchange components. You should read this section if you are writing an application that uses these components or if you are creating one of these components.

Importing and Exporting Movie Data

Your application starts a data import or export operation by calling the Movie Toolbox. There are several Movie Toolbox functions that allow you to specify a data import or data export component. For example, the `PasteHandleIntoMovie` and `ConvertFileToMovieFile` functions allow you to specify a movie data import component. The `PutMovieIntoTypedHandle` and `ConvertMovieToFile` functions allow you to specify a movie data export component. All of these functions select a component for you if you do not specify one yourself. For more information about these functions, see the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime*.

When you import data into a QuickTime movie, you can specify that the data be placed into a specific existing track in the movie, into a new track that is created by the movie data import component, or into one or more existing tracks (in this case, the component may create additional tracks, if necessary).

When you export data from a QuickTime movie, you can request data from a specific track or from the entire movie. In addition, you can specify a segment of the track or movie to be exported.

Configuring a Movie Data Exchange Component

You do not need to configure a movie data exchange component before you use it to convert data into or out of a QuickTime movie. These components are implemented in such a way that they can operate successfully using their own default configuration information. In fact, some data exchange components do not allow you to configure them. However, most data exchange components do support some or all of the configuration functions that are defined for components of this type.

If you are going to configure a data exchange component, you must do so before you start the data exchange operation. You must call the component directly in order to set the configuration—the Movie Toolbox does not do this for you. Use the functions described in “Configuring Movie Data Import Components” and “Configuring Movie Data Export Components,” as appropriate. Note that all of these functions are optional; that is, it is up to the developer of the component to decide whether or not to support a given configuration function. If the component does not support a function you have called, the component returns an error code of `badComponentSelector`.

Finding a Specific Movie Data Exchange Component

If you are going to specify a particular data exchange component to the Movie Toolbox, you must first open a connection to that component. Use the Component Manager’s `OpenDefaultComponent` or `OpenComponent` function to open a connection to a

Movie Data Exchange Components

movie data exchange component (see the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox* for more information about these functions). Before you can open that connection, however, you must find an appropriate movie data exchange component.

To find an appropriate data exchange component, you may need to use the Component Manager's `FindNextComponent` function. You specify the characteristics of the component you are seeking in a component description record—in particular, in the `componentType`, `componentSubtype`, `componentManufacturer`, and `componentFlags` fields.

Movie data import components have a component type value of `'eat '`, which is defined by the `MovieImportType` constant. Movie data export components have a type value of `'spit'`, which is defined by the `MovieExportType` constant.

Movie data exchange components use their component subtype and manufacturer values to indicate the type of data that they support. The subtype value indicates the type of data that these components can import or export. For example, movie data import components that convert text into QuickTime movie data have a component subtype value of `'TEXT'`. A single data exchange component may support only one data type.

The manufacturer field indicates the QuickTime media type that is supported by the component. For example, movie data export components that can read data from a sound media have a manufacturer value of `'soun'` (this value is defined by the `SoundMediaType` constant). If a data exchange component can work with more than one media type, it specifies a manufacturer value of 0.

In addition, these components use the `componentFlags` field to indicate more specific information about their capabilities. The following flags are currently defined:

```
enum {
    canMovieImportHandles      = 1,  /* can import from
                                     handles */
    canMovieImportFiles        = 2,  /* can import from files */
    hasMovieImportUserInterface = 4,  /* import has user
                                     interface */
    canMovieExportHandles      = 8,  /* can export to handles */
    canMovieExportFiles        = 16, /* can export to files */
    hasMovieExportUserInterface = 32, /* export has user
                                     interface */
    dontAutoFileMovieImport    = 64  /* turn off automatic file
                                     conversion */
};
```

Movie data import components use the first three flags to specify their capabilities. If a component can convert data from a handle, its `canMovieImportHandles` flag is set to 1. If it can work with files, its `canMovieImportFiles` flag is set to 1. Note that both of these flags may be set to 1 if a single component can work with both files and handles.

Movie Data Exchange Components

If a component provides a dialog box that allows the user to specify configuration information, the `hasMovieImportUserInterface` flag is set to 1. If a component does not support the automatic conversion of standard files to movies in an import component, set the `dontAutoFileMovieImport` flag to 1 (the default setting is 0).

Movie data export components use the other three flags in the same way.

Creating a Movie Data Exchange Component

This section discusses the details of creating a movie data exchange component. This section includes source code for two simple movie data exchange components.

You should consider creating a movie data import component if you have data that you would like to place in a QuickTime movie and there are not currently facilities for placing that type of data into a movie. Similarly, if you want to work with data from a QuickTime movie without using QuickTime, you might consider creating a movie data export component that can convert the data into a format your program can understand.

After reading this section, you should understand all of the special requirements of these components. The functional interface that your component must support is described in “Movie Data Exchange Components Reference” beginning on page 9-20. Note that a single component may support only import or export functions, not both.

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 component type values for movie data exchange components. You can use the following constants to specify this component type:

```
#define MovieImportType 'eat '      /* movie data import */
#define MovieExportType 'spit'     /* movie data export */
```

Apple has defined a functional interface for movie data exchange components. For information about the functions that your component must support, see “Movie Data Exchange Components Reference” beginning on page 9-20. You can use the following constants to refer to the request codes for each of the functions that your component must support:

```
enum {
    /* movie data import components */
    kMovieImportHandleSelect      = 1, /* import from handle */
    kMovieImportFileSelect       = 2, /* import from file */
    kMovieImportSetSampleDurationSelect = 3, /* set sample duration */
    kMovieImportSetSampleDescriptionSelect
                                   = 4, /* set sample description */
    kMovieImportSetMediaFileSelect = 5, /* set media file */
    kMovieImportSetDimensionsSelect = 6, /* set track dimensions */
```

Movie Data Exchange Components

```

kMovieImportSetChunkSizeSelect      = 7,  /* set chunk size */
kMovieImportSetProgressProcSelect   = 8,  /* set progress function */
kMovieImportSetAuxiliaryDataSelect  = 9,  /* set additional data */
kMovieImportSetFromScrapSelect      = 10, /* data from scrap */
kMovieImportDoUserDialogSelect      = 11, /* invoke user dialog box */
kMovieImportSetDurationSelect       = 12  /* set paste duration */

/* movie data export components */
kMovieExportToHandleSelect          = 128, /* export to handle */
kMovieExportToFileSelect            = 129, /* export to file */
kMovieExportDoUserDialogSelect      = 130, /* invoke user dialog box */
kMovieExportGetAuxiliaryDataSelect  = 131, /* get additional data */
kMovieExportSetProgressProcSelect   = 132 /* set progress function */
};

```

A Sample Movie Import Component

This section describes how to create a movie import component. First you implement the required functions. Then you instruct your component to obtain the movie data from a handle or a file. This section then supplies a sample program that implements a movie data exchange component that imports a Scrapbook file containing QuickDraw PICT images. (For details on QuickDraw PICT images, see the chapter “Basic QuickDraw” in *Inside Macintosh: Imaging*.)

Your movie data import component may provide a user dialog box. You may use this dialog box in any way that is appropriate for your component—for example, to obtain certain parameter information governing the import operation, such as the image-compression method.

In addition, the requesting application may use one or more of the configuration functions to establish parameters for the import operation.

You should not rely on any outside configuration information. Your component should work properly knowing only the source data and the target movie. The Movie Toolbox supplies this information to your component when it calls your `MovieImportHandle` function (described on page 9-21) or `MovieImportFile` function (described on page 9-24).

Your movie data import component may implement either one or both of these functions, which allow the Movie Toolbox to request that data be converted into a format for use in a QuickTime movie.

- If the data is to be imported from a handle, the `MovieImportHandle` function is used.
- If data is to be imported from a file, the `MovieImportFile` function is used.

Set the appropriate flags in your component’s `componentFlags` field to indicate which of these functions your component supports. Note that your component may support both functions.

Implementing the Required Import Component Functions

Listing 9-1 supplies a sample program that implements a movie data exchange component that imports a Scrapbook file containing QuickDraw PICT images. (For details on QuickDraw PICT images, see the chapter “Basic QuickDraw” in *Inside Macintosh: Imaging*.) The sample program also provides the dispatchers for the movie import component together with the required functions.

Listing 9-1 Implementing the required import functions

```
#define kMediaTimeScale 600

typedef struct {
    ComponentInstance self
    TimeValue          frameDuration;
} ImportScrapbookGlobalsRecord, **ImportScrapbookGlobals;

/* entry point for all Component Manager requests */
pascal ComponentResult ImportScrapbookDispatcher
    (ComponentParameters *params,
     Handle storage)
{
    OSErr err = badComponentSelector;
    ComponentFunction componentProc = 0;

    switch (params->what) {
        case kComponentOpenSelect:
            componentProc = ImportScrapbookOpen; break;

        case kComponentCloseSelect:
            componentProc = ImportScrapbookClose; break;

        case kComponentCanDoSelect:
            componentProc = ImportScrapbookCanDo; break;

        case kComponentVersionSelect:
            componentProc = ImportScrapbookVersion; break;

        case kMovieImportFileSelect:
            componentProc = ImportScrapbookFile; break;

        case kMovieImportSetSampleDurationSelect:
            componentProc = ImportScrapbookSetSampleDuration; break;

    }
}
```


Movie Data Exchange Components

```

        if (componentProc)
            err = CallComponentFunctionWithStorage (storage, params,
                                                    componentProc);

        return err;
    }

pascal ComponentResult ImportScrapbookCanDo
    (ImportScrapbookGlobals storage,
     short ftnNumber)
{
    switch (ftnNumber) {
        case kComponentOpenSelect:
        case kComponentCloseSelect:
        case kComponentCanDoSelect:
        case kComponentVersionSelect:
        case kMovieImportFileSelect:
        case kMovieImportSetSampleDurationSelect:
            return true;
        default:
            return false;
    }
}

pascal ComponentResult ImportScrapbookVersion
    (ImportScrapbookGlobals storage)
{
    return 0x00010001;
}

pascal ComponentResult ImportScrapbookOpen
    (ImportScrapbookGlobals storage,
     ComponentInstance self)
{
    storage = (ImportScrapbookGlobals) NewHandleClear
        (sizeof (ImportScrapbookGlobalsRecord));
    if (!storage) return MemError();
    (**storage).self = self;
    SetComponentInstanceStorage (self, (Handle)storage);
    return noErr;
}

pascal ComponentResult ImportScrapbookClose
    (ImportScrapbookGlobals storage,
     ComponentInstance self)

```

Movie Data Exchange Components

```

{
    if (storage) DisposeHandle((Handle)storage);
    return noErr;
}

```

Importing a Scrapbook File

Before the import operation begins, the client may set the duration of samples to be added by the movie data import component by calling the `MovieImportSetDuration` function (described on page 9-27).

The `MovieImportFile` function (described on page 9-24) performs the import operation. The tasks involved in importing the data include

- opening the source file
- retrieving the first sample in order to determine the track dimension
- creating a new track and media
- determining the frame duration
- setting up a sample description structure
- cycling through all the frames in the Scrapbook and adding them to the new media
- adding the new media to the track
- closing the source file

Listing 9-2 supplies an example in which a Scrapbook file is imported.

Listing 9-2 Importing a Scrapbook file

```

/* if this function is called, it provides a hint from the caller
as to the desired sample (frame) duration in the new media */

pascal ComponentResult ImportScrapbookSetSampleDuration
                                (ImportScrapbookGlobals storage,
                                 TimeValue duration,
                                 TimeScale scale)
{
    TimeRecord tr;
    tr.value.lo = duration;
    tr.value.hi = 0;
    tr.scale = 0;
    tr.base = nil;
    ConvertTimeScale (&tr, kMediaTimeScale);
    /* your new media will have a time scale of 600 */
    (**storage).frameDuration = tr.value.lo;
}

```

Movie Data Exchange Components

```

        return noErr;
    }
    pascal ComponentResult ImportScrapbookFile
        (ImportScrapbookGlobals storage,
         FSSpec *theFile, Movie theMovie,
         Track targetTrack, Track *usedTrack,
         TimeValue atTime,
         TimeValue *addedTime,
         long inFlags, long *outFlags)
    {
        OSErr err;
        short resRef = 0, saveRes = CurResFile();
        PicHandle thePict;
        Rect trackRect;
        short pageIndex = 0;
        Track newTrack = 0;
        Media newMedia;
        Boolean endMediaEdits = false;
        TimeValue frameDuration;
        SampleDescriptionHandle sampleDesc = 0;

        *outFlags = 0;
        if (inFlags & movieImportMustUseTrack)
            return invalidTrack;

        /* open source file */
        resRef = FSpOpenResFile (theFile, fsRdPerm);
        if (err = ResError()) goto bail;
        UseResFile(resRef);

        /* get the first PICT to determine the track size */
        thePict = (PicHandle)Get1IndResource ('PICT', 1);
        if (!thePict) {
            err = ResError();
            goto bail;
        }
        trackRect = (**thePict).picFrame;
        OffsetRect(&trackRect, -trackRect.left, -trackRect.top);
    }

```

Movie Data Exchange Components

```

/* create a track and PICT media */
newTrack = NewMovieTrack (theMovie, trackRect.right << 16,
                          trackRect.bottom << 16, kNoVolume);
if (err = GetMoviesError()) goto bail;
newMedia = NewTrackMedia (newTrack, 'PICT', kMediaTimeScale,
                          nil, 0);
if (err = GetMoviesError()) goto bail;
if (err = BeginMediaEdits (newMedia)) goto bail;
endMediaEdits = true;

/* determine the frame duration (check the hint you may
   have been called with) */
frameDuration = (**storage).frameDuration;
if (!frameDuration) frameDuration = kMediaTimeScale/5;
/* default is 1/5th second */

/* set up a simple sample description */
sampleDesc = (SampleDescriptionHandle) NewHandleClear
              (sizeof (SampleDescription));
(**sampleDesc).descSize = sizeof(SampleDescription);
(**sampleDesc).dataFormat = 'PICT';

/* cycle through all source frames and add them to the media */
do {
    Handle thePict;
    short resID = pageToMapIndex (++pageIndex,
                                  *GetResource ('SMAP', 0));

    if (resID == 0) break;
    thePict = Get1Resource ('PICT', resID);
    if (thePict == nil) continue; /* some pages may not
                                   contain a 'PICT' */

    err = AddMediaSample(newMedia, thePict, 0,
                        GetHandleSize (thePict),
                        frameDuration, sampleDesc, 1, 0, nil);

    ReleaseResource (thePict);
    DisposeHandle (thePict);
} while (!err);
if (err) goto bail;

```

Movie Data Exchange Components

```

    /* add the new media to the track */
    err = InsertMediaIntoTrack (newTrack, 0, 0,
                               GetMediaDuration (newMedia), kFix1);

bail:
    if (resRef) CloseResFile (resRef);
    if (endMediaEdits) EndMediaEdits (newMedia);
    if (err && newTrack) {
        DisposeMovieTrack (newTrack);
        newTrack = 0;
    }
    UseResFile (saveRes);
    if (sampleDesc) DisposeHandle ((Handle)sampleDesc);
    *usedTrack = newTrack;

    return err;
}

/* map from a Scrapbook page number to a resource ID */
short pageToMapIndex (short page, Ptr map)
{
    short mapIndex;
    for (mapIndex = 0; mapIndex < 256; mapIndex++)
        if (*map++ == page)
            return mapIndex | 0x8000;
    return 0;
}

```

A Sample Movie Export Component

As with movie data import components, the movie data export component should not rely on any configuration information beyond that which is supplied by the Movie Toolbox when it calls the `MovieExportToHandle` or `MovieExportToFile` function (described on page 9-35 and page 9-36, respectively).

This section provides an implementation of a movie data exchange component that exports a movie or movie's track to a PICS animation file.

Implementing the Required Export Component Functions

Listing 9-3 provides the component dispatchers for the movie export component together with the required functions.

Listing 9-3 Implementing the required export functions

```
typedef struct {
    ComponentInstance self;
} ExportPICSGlobalsRecord, *ExportPICSGlobals;

/* entry point for all Component Manager requests */
pascal ComponentResult ExportPICSDispatcher
    (ComponentParameters *params,
     Handle storage)
{
    OSErr err = badComponentSelector;
    ComponentFunction componentProc = 0;

    switch (params->what) {
        case kComponentOpenSelect:
            componentProc = ExportPICSOpen; break;
        case kComponentCloseSelect:
            componentProc = ExportPICSClose; break;
        case kComponentCanDoSelect:
            componentProc = ExportPICSCanDo; break;
        case kComponentVersionSelect:
            componentProc = ExportPICSVersion; break;
        case kMovieExportToFileSelect:
            componentProc = ExportPICSToFile; break;
    }
    if (componentProc)
        err = CallComponentFunctionWithStorage (storage, params,
                                                componentProc);

    return err;
}
```

Movie Data Exchange Components

```

pascal ComponentResult ExportPICSCanDo (ExportPICSGlobals store,
                                         short ftnNumber)
{
    switch (ftnNumber) {
        case kComponentOpenSelect:
        case kComponentCloseSelect:
        case kComponentCanDoSelect:
        case kComponentVersionSelect:
        case kMovieExportToFileSelect:
            return true;
            break;
        default:
            return false;
            break;
    }
}

pascal ComponentResult ExportPICSVersion (ExportPICSGlobals store)
{
    return 0x00010001;
}

pascal ComponentResult ExportPICSOpen (ExportPICSGlobals store,
                                       ComponentInstance self)
{
    OSErr err;

    store = (ExportPICSGlobals) NewPtrClear
        (sizeof(ExportPICSGlobalsRecord));
    if (err = MemError()) goto bail;
    store->self = self;

    SetComponentInstanceStorage(self, (Handle)store);

bail:
    return err;
}

pascal ComponentResult ExportPICSClose (ExportPICSGlobals store,
                                       ComponentInstance self)
{
    if (store) DisposPtr((Ptr)store);

    return noErr;
}

```

Exporting Data to a PICS File

To export data to a PICS file, your component must

- allow the Movie Toolbox to call the `MovieExportToFile` function in order to export movie data into a file
- read the data from the track or movie
- perform appropriate conversions on that data
- place the data into the specified file (the file's type corresponds to the component subtype of your movie data export component)

Listing 9-4 provides an implementation of these tasks in a movie export component. The `ExportPICSToFile` function performs the export operation by opening the resource fork of the PICS file and cycling through the movie time segment, adding a frame to the PICS file.

Listing 9-4 Exporting a frame of movie data to a PICS file

```
pascal ComponentResult ExportPICSToFile (ExportPICSGlobals store,
                                         const FSSpec *theFile,
                                         Movie m,
                                         Track onlyThisTrack,
                                         TimeValue startTime,
                                         TimeValue duration)
{
    OSErr err = noErr;
    short resRef = 0;
    short saveResRef = CurResFile();
    short resID = 128;
    PicHandle thePict = nil;

    /* open the resource fork of the PICS file
       (the caller is responsible for creating the file) */
    resRef = FSpOpenResFile (theFile, fsRdWrPerm);
    if (err = ResError()) goto bail;

    UseResFile(resRef);

    /* cycle through the movie time segment you were given */
    while (startTime < duration) {
        Byte c = 0;
```


Movie Data Exchange Components

```

        if (onlyThisTrack)
            thePict = GetTrackPict (onlyThisTrack, startTime);
        else
            thePict = GetMoviePict(m, startTime);
        if (!thePict) continue;

        /* add a frame to the PICS file */
        AddResource ((Handle)thePict, 'PICT', resID++, &c);
        err = ResError();
        WriteResource ((Handle)thePict);
        DetachResource ((Handle)thePict);
        KillPicture (thePict);
        thePict = nil;
        if (err) break;

        /* find the time of the next frame */
        do {
            TimeValue nextTime;
            if (onlyThisTrack)
                GetTrackNextInterestingTime (onlyThisTrack,
                                                nextTimeMediaSample, startTime,
                                                kFix1, &nextTime, nil);
            else {
                OSType mediaType = VisualMediaCharacteristic;

                GetMovieNextInterestingTime (m, nextTimeMediaSample,
                                                1, &mediaType,
                                                startTime, kFix1,
                                                &nextTime, nil);
            }

            if (GetMoviesError ()) goto bail;
            if (nextTime != startTime) {
                startTime = nextTime;
                break;
            }
        } while (++startTime < duration);
    }

bail:
    if (thePict) KillPicture (thePict);
    if (resRef) CloseResFile (resRef);
    UseResFile (saveResRef);
    return err;
}

```

Movie Data Exchange Components Reference

This section describes the functions that your movie data exchange component may support. Many of these functions are optional—your component should support only those functions that are appropriate to it.

This section is divided into the following topics:

- “Importing Movie Data” discusses the functions that allow the Movie Toolbox to import movie data using the services of a movie data import component.
- “Configuring Movie Data Import Components” describes the functions that allow applications to configure a movie data import component prior to importing movie data.
- “Exporting Movie Data” tells you about the functions that allow the Movie Toolbox to export movie data using the services of a movie data export component.
- “Configuring Movie Data Export Components” provides information about the functions that allow applications to configure a movie data export component prior to exporting movie data.

Note

All of the functions described in “Configuring Movie Data Import Components” and “Configuring Movie Data Export Components” are optional. Your import or export component must be able to work properly if none of these functions is called. ♦

Importing Movie Data

Movie data import components may provide one or two functions that allow the Movie Toolbox to request a data conversion operation. The `MovieImportHandle` function instructs your component to retrieve the data that is to be imported from a specified handle. The `MovieImportFile` function instructs you to retrieve the data from a file. You should set the appropriate flags in your component’s `componentFlags` field to indicate which of these functions your component supports. Note that your component may support both functions.

Before the Movie Toolbox calls one of these functions, a requesting application may call one or more of your component’s configuration functions (see “Configuring Movie Data Import Components” beginning on page 9-26 for more information about these functions). However, your component should work properly even if none of these configuration functions is called.

MovieImportHandle

The `MovieImportHandle` function allows the Movie Toolbox to import data from a handle, using your movie data import component.

```
pascal ComponentResult MovieImportHandle (ComponentInstance ci,
                                           Handle dataH,
                                           Movie theMovie,
                                           Track targetTrack,
                                           Track *usedTrack,
                                           TimeValue atTime,
                                           TimeValue *addedDuration,
                                           long inFlags,
                                           long *outFlags);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your movie data import component.
<code>dataH</code>	Contains a handle to the data that is to be imported into the movie identified by the parameter <code>theMovie</code> . The data contained in this handle has a data type value that corresponds to your component's subtype value. Your component is not responsible for disposing of this handle.
<code>theMovie</code>	Identifies the movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie, or to obtain information about the movie.
<code>targetTrack</code>	Identifies the track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the <code>movieImportMustUseTrack</code> flag in the <code>inFlags</code> parameter is set to 1.
<code>usedTrack</code>	Contains a pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.
<code>atTime</code>	Specifies the time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.
<code>addedDuration</code>	Contains a pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.

Movie Data Exchange Components

<code>inFlags</code>	Specifies control information governing the import operation. The following flags are defined:
<code>movieImportCreateTrack</code>	<p>Indicates that your component should create a new track to receive the imported data. You must create a track whose type value corresponds to the media type that you have specified in your component's manufacturer code. You should return the track identifier of this new track in the field referred to by the <code>usedTrack</code> parameter, unless you create more than one track. If you create more than one track, be sure to set the <code>movieImportResultUsedMultipleTracks</code> flag (in the field referred to by the <code>outFlags</code> parameter) to 1.</p> <p>If the <code>movieImportCreateTrack</code> flag is set to 1, then the <code>movieImportMustUseTrack</code> flag is set to 0.</p>
<code>movieImportMustUseTrack</code>	<p>Indicates that your component must use an existing track. That track is identified by the <code>targetTrack</code> parameter. If you create more than one track, be sure to set the <code>movieImportResultUsedMultipleTracks</code> flag (in the field referred to by the <code>outFlags</code> parameter) to 1.</p> <p>If the <code>movieImportMustUseTrack</code> flag is set to 1, then the <code>movieImportCreateTrack</code> flag is set to 0.</p> <p>If both the <code>movieImportCreateTrack</code> and <code>movieImportMustUseTrack</code> flags are set to 0, then you are free to use any existing tracks in the movie or to create a new track (or tracks) as needed.</p>
<code>movieImportInParallel</code>	<p>Indicates whether you are to perform an insert operation or a paste operation. If this flag is set to 0, then you should insert the imported data into the target track. If this flag is set to 1, then you should add the imported data to the track, overwriting preexisting open space currently in the track. Note that an application may use the <code>MovieImportSetDuration</code> function (described on page 9-27) to control the amount of data you paste into a movie.</p> <p>If the <code>movieImportMustUseTrack</code> flag is set to 1, then you should use the track specified by the <code>targetTrack</code> parameter. If this is not possible, return an appropriate Movie Toolbox result code.</p>

Movie Data Exchange Components

`outFlags` Contains a pointer to a field that is to receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete. The following flags are defined:

`movieImportResultUsedMultipleTracks`

Indicates that your component modified more than one track in the movie. Set this flag to 1 if your component places imported data into more than one track. In this case, you do not need to update the field referred to by the `usedTrack` parameter.

`movieImportInParallel`

Indicates whether you performed an insert operation or a paste operation. Set this flag to 0 if you inserted the imported data into the target track. Set this flag to 1 if you added the imported data to the track, overwriting preexisting open space currently in the track.

DESCRIPTION

The Movie Toolbox calls the `MovieImportHandle` function in order to import movie data from a handle. The data stored in the handle has a data type that corresponds to the component subtype of your movie data import component. Your component must read the data from the supplied handle, perform appropriate conversions on that data, and place the data into the movie.

If your component can accept data from a handle, be sure to set the `canMovieImportHandles` flag in your component's `componentFlags` field.

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

RESULT CODES

`invalidTrack` -2009 Specified track cannot receive imported data
Other appropriate Movie Toolbox result codes

SEE ALSO

The Movie Toolbox uses the `MovieImportFile` function to import data from a file; this function is described next.

MovieImportFile

The `MovieImportFile` function allows the Movie Toolbox to import data from a file, using your movie data import component.

```
pascal ComponentResult MovieImportFile (ComponentInstance ci,
                                         const FSSpec *theFile,
                                         Movie theMovie,
                                         Track targetTrack,
                                         Track *usedTrack,
                                         TimeValue atTime,
                                         TimeValue *addedDuration,
                                         long inFlags,
                                         long *outFlags);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your movie data import component.
<code>theFile</code>	Contains a pointer to the file that contains the data that is to be imported into the movie. This file's type value corresponds to your component's subtype value.
<code>theMovie</code>	Identifies the movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie or to obtain information about the movie.
<code>targetTrack</code>	Identifies the track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the <code>movieImportMustUseTrack</code> flag in the <code>inFlags</code> parameter is set to 1.
<code>usedTrack</code>	Contains a pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.
<code>atTime</code>	Specifies the time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.
<code>addedDuration</code>	Contains a pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.
<code>inFlags</code>	Specifies control information governing the import operation. The following flags are defined: <div> <div><code>movieImportCreateTrack</code></div> <div>Indicates that your component should create a new track to receive the imported data. You must create a track whose type value corresponds to the media type you have specified in your component's manufacturer code. You</div> </div>

should return the track identifier of this new track in the field referred to by the `usedTrack` parameter, unless you create more than one track. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag (in the field referred to by the `outFlags` parameter) to 1.

If the `MovieImportCreateTrack` flag is set to 1, then the `movieImportMustUseTrack` flag is set to 0.

`movieImportMustUseTrack`

Indicates that your component must use an existing track. That track is identified by the `targetTrack` parameter. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag (in the field referred to by the `outFlags` parameter) to 1.

If the `movieImportMustUseTrack` flag is set to 1, then the `movieImportCreateTrack` flag is set to 0.

If both the `movieImportCreateTrack` and `movieImportMustUseTrack` flags are set to 0, then you are free to use any existing tracks in the movie, or to create a new track (or tracks) as needed.

`movieImportInParallel`

Indicates whether you are to perform an insert operation or a paste operation. If this flag is set to 0, then you should insert the imported data into the target track. If this flag is set to 1, then you should add the imported data to the track, overwriting the preexisting open space currently in the track. Note that an application may use the `MovieImportSetDuration` function to control the amount of data you paste into a movie.

If the `movieImportMustUseTrack` flag is set to 1, then you should use the track specified by the `targetTrack` parameter. If this is not possible, return an appropriate Movie Toolbox result code.

`outFlags` Identifies a field that is to receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete. The following flags are defined:

`movieImportResultUsedMultipleTracks`

Indicates that your component modified more than one track in the movie. Set this flag to 1 if your component places imported data into more than one track. In this case, you do not need to update the field referred to by the `usedTrack` parameter.

`movieImportInParallel`

Indicates whether you performed an insert operation or a paste operation. Set this flag to 0 if you inserted the imported data into the target track. Set this flag to 1 if you added the imported data to the track, overwriting preexisting open space currently in the track.

DESCRIPTION

The Movie Toolbox calls the `MovieImportFile` function in order to import movie data from a file. The file's type corresponds to the component subtype of your movie data import component. Your component must read the data from the supplied file, perform appropriate conversions on that data, and place the data into the movie.

If your component can accept data from a file, be sure to set the `canMovieImportFiles` flag in your component's `componentFlags` field.

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

RESULT CODES

`invalidTrack` -2009 Specified track cannot receive imported data
Other appropriate Movie Toolbox result codes

SEE ALSO

The Movie Toolbox uses the `MovieImportHandle` function to import data from a handle; this function is described on page 9-21.

Configuring Movie Data Import Components

Your component may provide one or more configuration functions. These functions allow applications to configure your component before the Movie Toolbox calls your component to start the import process. Note that applications may call these functions directly.

All of these functions are optional. If your component receives a request that it does not support, you should return the `badComponentSelector` error code. In addition, your component should work properly even if none of these functions is called.

These functions address a variety of configuration issues. The `MovieImportSetSampleDuration` function allows an application to set your component's sample duration. Use the `MovieImportSetDuration` function to control the duration of the imported data. Applications can use the `MovieImportSetDimensions` function to specify the spatial dimensions of a new track. Use the `MovieImportSetSampleDescription` function to supply a sample description structure to your movie data import component.

The `MovieImportSetMediaFile` function allows applications to direct your component's output to a specific media file. Applications can provide additional data to your component by calling the `MovieImportSetAuxiliaryData` function. The `MovieImportSetChunkSize` function allows applications to control the chunk size in the new media. Applications can inform you that the source data came from the scrap by calling your `MovieImportSetFromScrap` function.

Applications can specify a progress function for use by your component by calling the `MovieImportSetProgressProc` function.

Applications can instruct your component to display its user dialog box by calling the `MovieImportDoUserDialog` function.

MovieImportSetDuration

The `MovieImportSetDuration` function allows an application to control the duration of the data that your component pastes into the target movie.

```
pascal ComponentResult MovieImportSetDuration
                                (ComponentInstance ci,
                                TimeValue duration);
```

ci	Identifies the application's connection to your movie data import component.
duration	Specifies the duration in the movie's time scale. If this parameter is set to 0, then you may paste any amount of movie data that is appropriate for the data to be imported.

DESCRIPTION

Applications may use the `MovieImportSetDuration` function to set the duration of the data to be pasted by your movie data import component. This duration is expressed in the movie's time scale.

If your component supports paste operations (that is, your component allows the application to set the `movieImportInParallel` flag to 1 with the `MovieImportHandle` or `MovieImportFile` function), then you must support this function. If an application calls this function and sets a duration limit, you must abide by that limit. This function is not valid for insert operations (where the `movieImportInParallel` flag is set to 0).

RESULT CODE

badComponentSelector	0x80008002	Function not supported
----------------------	------------	------------------------

MovieImportSetSampleDuration

The `MovieImportSetSampleDuration` function allows an application to set the sample duration for new samples to be created with your component.

```
pascal ComponentResult MovieImportSetSampleDuration
    (ComponentInstance ci,
     TimeValue duration,
     TimeScale scale);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>duration</code>	Specifies the sample duration in units specified by the <code>scale</code> parameter.
<code>scale</code>	Specifies the time scale for the duration value. This may be any arbitrary time scale; that is, it may not correspond to the movie's time scale. You should convert this time scale to the movie's time scale before using the duration value, using the Movie Toolbox's <code>ConvertTimeScale</code> function.

DESCRIPTION

Applications may use the `MovieImportSetSampleDuration` function to set the duration of samples to be added by your movie data import component. This duration is expressed in an arbitrary time scale.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

MovieImportSetSampleDescription

The `MovieImportSetSampleDescription` function allows an application to provide a sample description to your movie data import component.

```
pascal ComponentResult MovieImportSetSampleDescription
    (ComponentInstance ci,
     SampleDescriptionHandle desc,
     OSType mediaType);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
-----------------	--

desc	Contains a handle to a sample description. Your component must not dispose of this handle. If you want to save any data from the sample description, be sure to copy it at this time.
mediaType	Specifies the type of sample description referred to by the desc parameter. If the desc parameter refers to an image description structure, this parameter is set to VideoMediaType ('vide'); for sound description structures, this parameter is set to SoundMediaType ('soun').

DESCRIPTION

Applications may use the `MovieImportSetSampleDescription` function to supply a sample description to your movie data import component. This can be useful in cases where your component must transform the data before adding it to the movie's media. For example, your component may be responsible for adding image data to a movie. In this case, you may allow applications to specify image-compression parameters by supplying a formatted image description structure.

RESULT CODE

badComponentSelector	0x80008002	Function not supported
----------------------	------------	------------------------

MovieImportSetMediaFile

The `MovieImportSetMediaFile` function allows an application to specify a media file that is to receive the imported movie data.

```
pascal ComponentResult MovieImportSetMediaFile
                                (ComponentInstance ci,
                                AliasHandle alias);
```

ci	Identifies the application's connection to your movie data import component.
alias	Identifies the media file that is to receive the imported movie data. Your component must make a copy of this parameter. You should not dispose of it.

DESCRIPTION

Applications may use the `MovieImportSetMediaFile` function to specify a destination media file for imported movie data. By default, your movie data import component should add new data to an existing media file that is associated with the movie. However, you may choose to allow applications to specify an alternative destination file. This can be useful when your component is importing data into a new

Movie Data Exchange Components

track. In this case, the application can use this function to tell your component where the media's data should reside.

RESULT CODE

badComponentSelector 0x80008002 Function not supported

MovieImportSetDimensions

The `MovieImportSetDimensions` function allows an application to specify a new track's spatial dimensions.

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

<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>width</code>	Indicates the width, in pixels, of the track rectangle. This parameter, along with the <code>height</code> parameter, specifies a rectangle that surrounds the image that is to be displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle, and it 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 it is expressed as a fixed-point number.

DESCRIPTION

Applications may use this function to specify the spatial dimensions of a new track. Although your movie data import component may not change the spatial characteristics of an existing track, if you are importing image data into a new track, you may choose to allow applications to specify the spatial characteristics of the new track.

If you want to change the track's matrix, use the Movie Toolbox's `SetTrackMatrix` function after performing the import operation.

RESULT CODE

badComponentSelector 0x80008002 Function not supported

MovieImportSetChunkSize

The `MovieImportSetChunkSize` function allows an application to specify the amount of data your component works with at a time.

```
pascal ComponentResult MovieImportSetChunkSize
                                (ComponentInstance ci,
                                 long chunkSize);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>chunkSize</code>	Specifies the number of seconds of data your movie data import component places into each chunk of movie data. This parameter may not be set to a value less than 1.

DESCRIPTION

The chunk size controls the amount of data in each of a media's data chunks (for more information about data chunks in a media, see the chapter "QuickTime Movie Format" in *Inside Macintosh: QuickTime*). Generally, your component should determine a reasonable default chunk size, based on the type of data you are importing. However, you may choose to allow applications to override your default value—this can be especially useful for sound data, where the chunk size affects the quality of sound playback.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

MovieImportSetProgressProc

The `MovieImportSetProgressProc` function allows an application to assign a movie progress function.

```
pascal ComponentResult MovieImportSetProgressProc
                                (ComponentInstance ci,
                                 MovieProgressProcPtr proc,
                                 long refcon);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
-----------------	--

Movie Data Exchange Components

<code>proc</code>	Contains a pointer to the application's movie progress function. See the chapter "Movie Toolbox" in <i>Inside Macintosh: QuickTime</i> for a complete description of the interface supported by movie progress functions. If this parameter is set to <code>nil</code> , the application is removing its progress function. In this case, your component should stop calling the progress function.
<code>refcon</code>	Specifies a reference constant. Your component should pass this constant back to the application's progress function whenever you call that function.

DESCRIPTION

Some data import operations may be time consuming, and application developers may therefore choose to display progress information to the user. Your component provides this information to an application's progress function. As your component processes an import request, you should call the progress function occasionally in order to report on the progress of the operation. Use an operation code value of `progressOpImportMovie`. The application can then present this information to the user.

These progress functions must support the same interface as Movie Toolbox progress functions. That interface is discussed in the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime*. Note that this interface not only allows you to report progress to the application, but also allows the application to cancel the request.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

MovieImportSetAuxiliaryData

The `MovieImportSetAuxiliaryData` function allows an application to provide additional data to your component. Your component can then use this data during the data import process.

```
pascal ComponentResult MovieImportSetAuxiliaryData
                                (ComponentInstance ci,
                                 Handle data,
                                 OSType handleType);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>data</code>	Contains a handle to the additional data. Your component should not dispose of this handle. Be sure to copy any data you need to keep.
<code>handleType</code>	Identifies the type of data in the specified handle.

DESCRIPTION

The `MovieImportSetAuxiliaryData` function allows your component to accept additional data for use during the data import process. Your component may use this data in any way that is appropriate for a given import operation. For example, if your component imports data stored in 'TEXT' handles, you might choose to accept style information for that text. An application could provide that style information in a 'styl' handle supplied to your component by calling this function.

Your component should expect the application to call this function before the import process begins.

RESULT CODES

<code>unsupportedAuxiliaryImportData</code>	<code>-2057</code>	Cannot work with specified handle type
<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported

MovieImportSetFromScrap

The `MovieImportSetFromScrap` function allows an application to indicate that the source data resides on the scrap.

<pre>pascal ComponentResult MovieImportSetFromScrap (ComponentInstance ci, Boolean fromScrap);</pre>	
<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>fromScrap</code>	Indicates whether or not the source data resides on the scrap. This parameter is set to <code>true</code> if the data originated on the scrap; otherwise, the parameter is set to <code>false</code> .

DESCRIPTION

The `MovieImportSetFromScrap` function allows an application to indicate that the data to be imported originated on the scrap. In some cases, your component may be able to use this information during the import process. For example, you may establish the convention that additional data that is pertinent to an import operation should be stored on the scrap along with the data to be imported. Your component can then look in the scrap for the additional data.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

MovieImportDoUserDialog

The `MovieImportDoUserDialog` function allows an application to request that your component display its user dialog box.

```
pascal ComponentResult MovieImportDoUserDialog
    (ComponentInstance ci,
     const FSSpec *theFile,
     Handle theData, Boolean *canceled);
```

<code>ci</code>	Identifies the application's connection to your movie data import component.
<code>theFile</code>	Contains a pointer to a valid file specification. If the import request pertains to a file, the application must specify the source file with this parameter and set the parameter <code>theData</code> to <code>nil</code> . If the request is for a handle, this parameter is set to <code>nil</code> .
<code>theData</code>	Contains a handle to the data to be imported. If the import request pertains to a handle, the application must specify the source of the data with this parameter, and set the parameter <code>theFile</code> to <code>nil</code> . If the request is for a file, this parameter is set to <code>nil</code> .
<code>canceled</code>	Contains a pointer to a Boolean value. Your component should set this Boolean value to reflect whether the user cancels the dialog box. If the user cancels the dialog box, set the Boolean value to <code>true</code> . Otherwise, set it to <code>false</code> .

DESCRIPTION

Your movie data import component may support a user dialog box that allows the user to configure an import operation. For components that support such a dialog box, the `MovieImportDoUserDialog` function allows an application to tell you when to display the dialog box to the user.

If your component supports a user dialog box, be sure to set the `hasMovieImportUserInterface` flag in your component's `componentFlags` field.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

Exporting Movie Data

Movie data export components may provide one or two functions that allow the Movie Toolbox to request a data conversion operation. The `MovieExportToHandle` function instructs your component to place the converted data into a specified handle. The `MovieExportToFile` function instructs you to put the data into a file. You should set the appropriate flags in your component's `componentFlags` field to indicate which of

these functions your component supports. Note that your component may support both functions.

Before the Movie Toolbox calls one of these functions, a requesting application may call one or more of your component’s configuration functions (see “Configuring Movie Data Export Components” beginning on page 9-37 for more information about these functions). However, your component should work properly even if none of these configuration functions is called.

MovieExportToHandle

The `MovieExportToHandle` function allows the Movie Toolbox to export data from a movie, using your movie data export component.

<pre>pascal ComponentResult MovieExportToHandle (ComponentInstance ci, Handle dataH, Movie theMovie, Track onlyThisTrack, TimeValue startTime, TimeValue duration);</pre>	
<code>ci</code>	Identifies the Movie Toolbox’s connection to your movie data export component.
<code>dataH</code>	Handle to be filled with the converted movie data. Your component must write data into this handle that corresponds to your component’s subtype value. Your component should resize this handle as appropriate.
<code>theMovie</code>	Identifies the movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.
<code>onlyThisTrack</code>	Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.
<code>startTime</code>	Specifies the starting point of the track or movie segment to be converted. This time value is expressed in the movie’s time coordinate system.
<code>duration</code>	Specifies the duration of the track or movie segment to be converted. This duration value is expressed in the movie’s time coordinate system.

DESCRIPTION

The Movie Toolbox calls the `MovieExportToHandle` function in order to export movie data into a handle. Your component must read the data from the specified movie or track, perform appropriate conversions on that data, and place the data into the handle.

Movie Data Exchange Components

The data stored in the handle must have a data type that corresponds to the component subtype of your movie data export component.

If your component can write data to a handle, be sure to set the `canMovieExportHandles` flag in your component's `componentFlags` field.

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

RESULT CODES

`invalidTrack` -2009 Specified track cannot be converted

Other appropriate Movie Toolbox result codes

SEE ALSO

The Movie Toolbox uses the `MovieExportToFile` function to export data to a file; this function is described next.

MovieExportToFile

The `MovieExportToFile` function allows the Movie Toolbox to export data to a file, using your movie data export component.

```
pascal ComponentResult MovieExportToFile (ComponentInstance ci,
                                           const FSSpec *theFile,
                                           Movie theMovie,
                                           Track onlyThisTrack,
                                           TimeValue startTime,
                                           TimeValue duration);
```

<code>ci</code>	Identifies the Movie Toolbox's connection to your movie data import component.
<code>theFile</code>	Contains a pointer to the file that is to receive the converted movie data. This file's type value corresponds to your component's subtype value.
<code>theMovie</code>	Identifies the movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.
<code>onlyThisTrack</code>	Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.
<code>startTime</code>	Specifies the starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration Specifies the duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

DESCRIPTION

The Movie Toolbox calls the `MovieExportToFile` function in order to export movie data into a file. Your component must read the data from the track or movie, perform appropriate conversions on that data, and place the data into the specified file. The file's type corresponds to the component subtype of your movie data export component.

Note that the requesting program or toolbox must create the destination file before calling this function. Furthermore, your component may not destroy any data in the destination file. If you cannot add data to the specified file, return an appropriate error.

If your component can write data to a file, be sure to set the `canMovieExportFiles` flag in your component's `componentFlags` field.

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

RESULT CODES

`invalidTrack` -2009 Specified track cannot be converted
Other appropriate Movie Toolbox result codes

SEE ALSO

The Movie Toolbox uses the `MovieExportToHandle` function to export data to a file; this function is described in the previous section.

Configuring Movie Data Export Components

Your component may provide one or more configuration functions. These functions allow applications to configure your component before the Movie Toolbox calls your component to start the export process. Note that applications may call these functions directly.

All of these functions are optional. If your component receives a request that it does not support, you should return the `badComponentSelector` error code. In addition, your component should work properly even if none of these functions is called.

These functions address a variety of configuration issues. Applications can retrieve additional data from your component by calling the `MovieExportGetAuxiliaryData` function.

Applications can specify a progress function for use by your component by calling the `MovieExportSetProgressProc` function.

Applications can instruct your component to display its user dialog box by calling the `MovieExportDoUserDialog` function.

MovieExportSetProgressProc

The `MovieExportSetProgressProc` function allows an application to assign a movie progress function.

```
pascal ComponentResult MovieExportSetProgressProc
                                (ComponentInstance ci,
                                MovieProgressProcPtr proc,
                                long refcon);
```

<code>ci</code>	Identifies the application's connection to your movie data export component.
<code>proc</code>	Contains a pointer to the application's movie progress function. See the chapter "Movie Toolbox" in <i>Inside Macintosh: QuickTime</i> for a complete description of the interface supported by movie progress functions. If this parameter is set to <code>nil</code> , the application is removing its progress function. In this case, your component should stop calling the progress function.
<code>refcon</code>	Specifies a reference constant. Your component should pass this constant back to the application's progress function whenever you call that function.

DESCRIPTION

Some data export operations may be time-consuming, and application developers may therefore choose to display progress information to the user. Your component provides this information to an application's progress function. As your component processes an export request, you should call the progress function occasionally in order to report on the progress of the operation. Use a progress code of `progressOpExportMovie`. The application can then present this information to the user.

These progress functions must support the same interface as Movie Toolbox progress functions. That interface is discussed in the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime*. Note that this interface not only allows you to report progress to the application, but also allows the application to cancel the request.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

MovieExportGetAuxiliaryData

The `MovieExportGetAuxiliaryData` function allows an application to retrieve additional data from your component. This additional data may be created during the data export process.

```
pascal ComponentResult MovieExportGetAuxiliaryData
                                     (ComponentInstance ci,
                                     Handle dataH,
                                     OSType *handleType);
```

ci	Identifies the application's connection to your movie data export component.
data	Contains a handle that is to be filled with the additional data. Your component should resize this handle as appropriate. Your component is not responsible for disposing of this handle.
handleType	Contains a pointer to the type of data you place in the handle specified by the data parameter.

DESCRIPTION

The `MovieExportGetAuxiliaryData` function allows an application to retrieve additional data that is generated during the data export process. The application may then use the data as appropriate. Your component may create this data in cases where the target data type cannot accommodate all of the converted data. For example, if your component exports data into 'TEXT' handles or files, you might choose to preserve associated style information for that text. However, 'TEXT' resources cannot store that style information. You could save that style information in a 'styl' handle and allow an application to retrieve it after the conversion.

Your component should expect the application to call this function after the export process ends.

RESULT CODE

badComponentSelector	0x80008002	Function not supported
----------------------	------------	------------------------

MovieExportDoUserDialog

The `MovieExportDoUserDialog` function allows an application to request that your component display its user dialog box.

```
pascal ComponentResult MovieExportDoUserDialog
    (ComponentInstance ci,
     const FSSpec *theFile,
     Handle theData,
     Boolean *canceled);
```

<code>ci</code>	Identifies the application's connection to your movie data export component.
<code>theFile</code>	Contains a pointer to a valid file specification. If the export request pertains to a file, the application must specify the destination file with this parameter and set the parameter <code>theData</code> to <code>nil</code> . If the request is for a handle, this parameter is set to <code>nil</code> .
<code>theData</code>	Contains a handle to receive the converted data. If the export request pertains to a handle, the application must specify the destination handle with this parameter, and set the parameter <code>theFile</code> to <code>nil</code> . If the request is for a file, this parameter is set to <code>nil</code> .
<code>canceled</code>	Contains a pointer to a Boolean value. Your component should set this Boolean value to reflect whether the user cancels the dialog box. If the user cancels the dialog box, set the Boolean value to <code>true</code> . Otherwise, set it to <code>false</code> .

DESCRIPTION

Your movie data export component may support a user dialog box that allows the user to configure an export operation. For components that support such a dialog box, the `MovieExportDoUserDialog` function allows an application to tell you when to display the dialog box to the user.

If your component supports a user dialog box, be sure to set the `hasMovieExportUserInterface` flag in your component's `componentFlags` field.

RESULT CODE

<code>badComponentSelector</code>	<code>0x80008002</code>	Function not supported
-----------------------------------	-------------------------	------------------------

Summary of Movie Data Exchange Components

C Summary

Constants

```

/* component type values */
#define MovieImportType 'eat '      /* movie data import */
#define MovieExportType 'spit'     /* movie data export */

/* componentFlags values for movie import and movie export components */
enum {
    canMovieImportHandles          = 1,  /* can import from handles */
    canMovieImportFiles            = 2,  /* can import from files */
    hasMovieImportUserInterface    = 4,  /* import has user interface */
    canMovieExportHandles          = 8,  /* can export to handles */
    canMovieExportFiles            = 16, /* can export to files */
    hasMovieExportUserInterface    = 32, /* export has user interface */
    dontAutoFileMovieImport        = 64  /* do not automatically import
                                         movie files */
};

/* flags for MovieImportHandle and MovieImportFile */
enum {
    movieImportCreateTrack          = 1,  /* create a new track */
    movieImportInParallel           = 2,  /* paste imported data */
    movieImportMustUseTrack         = 4   /* use specified track */
};

enum {
    movieImportResultUsedMultipleTracks = 8, /* component used several
                                         tracks */
};

enum {
    /* movie data import components */
    kMovieImportHandleSelect        = 1,  /* import from handle */
    kMovieImportFileSelect          = 2,  /* import from file */
    kMovieImportSetSampleDurationSelect = 3, /* set sample duration */
};

```

Movie Data Exchange Components

```

kMovieImportSetSampleDescriptionSelect = 4, /* set sample description */
kMovieImportSetMediaFileSelect         = 5, /* set media file */
kMovieImportSetDimensionsSelect        = 6, /* set track dimensions */
kMovieImportSetChunkSizeSelect          = 7, /* set chunk size */
kMovieImportSetProgressProcSelect       = 8, /* set progress func */
kMovieImportSetAuxiliaryDataSelect      = 9, /* set additional data */
kMovieImportSetFromScrapSelect          = 10, /* data from scrap */
kMovieImportDoUserDialogSelect          = 11, /* invoke user dialog */
kMovieImportSetDurationSelect           = 12 /* set paste duration */

/* movie data export components */
kMovieExportToHandleSelect              = 128, /* export to handle */
kMovieExportToFileSelect                = 129, /* export to file */
kMovieExportDoUserDialogSelect          = 130, /* invoke user dialog */
kMovieExportGetAuxiliaryDataSelect      = 131, /* get additional data */
kMovieExportSetProgressProcSelect       = 132 /* set progress function */
};

```

Data Type

```
typedef ComponentInstance MovieImportComponent, MovieExportComponent;
```

Functions

Importing Movie Data

```

pascal ComponentResult MovieImportHandle
    (ComponentInstance ci,
     Handle dataH, Movie theMovie,
     Track targetTrack, Track *usedTrack,
     TimeValue atTime, TimeValue *addedDuration,
     long inFlags, long *outFlags);

pascal ComponentResult MovieImportFile
    (ComponentInstance ci,
     const FSSpec *theFile, Movie theMovie,
     Track targetTrack, Track *usedTrack,
     TimeValue atTime, TimeValue *addedDuration,
     long inFlags, long *outFlags);

```


Configuring Movie Data Import Components

```

pascal ComponentResult MovieImportSetDuration
    (ComponentInstance ci, TimeValue duration);

pascal ComponentResult MovieImportSetSampleDuration
    (ComponentInstance ci, TimeValue duration,
     TimeScale scale);

pascal ComponentResult MovieImportSetSampleDescription
    (ComponentInstance ci,
     SampleDescriptionHandle desc,
     OSType mediaType);

pascal ComponentResult MovieImportSetMediaFile
    (ComponentInstance ci, AliasHandle alias);

pascal ComponentResult MovieImportSetDimensions
    (ComponentInstance ci,
     Fixed width, Fixed height);

pascal ComponentResult MovieImportSetChunkSize
    (ComponentInstance ci, long chunkSize);

pascal ComponentResult MovieImportSetProgressProc
    (ComponentInstance ci,
     MovieProgressProcPtr proc, long refcon);

pascal ComponentResult MovieImportSetAuxiliaryData
    (ComponentInstance ci,
     Handle data, OSType handleType);

pascal ComponentResult MovieImportSetFromScrap
    (ComponentInstance ci, Boolean fromScrap);

pascal ComponentResult MovieImportDoUserDialog
    (ComponentInstance ci, const FSSpec *theFile,
     Handle theData, Boolean *canceled);

```

Exporting Movie Data

```

pascal ComponentResult MovieExportToHandle
    (ComponentInstance ci, Handle dataH,
     Movie theMovie, Track onlyThisTrack,
     TimeValue startTime, TimeValue duration);

pascal ComponentResult MovieExportToFile
    (ComponentInstance ci,
     const FSSpec *theFile, Movie theMovie,
     Track onlyThisTrack, TimeValue startTime,
     TimeValue duration);

```

Configuring Movie Data Export Components

```
pascal ComponentResult MovieExportSetProgressProc
    (ComponentInstance ci,
     MovieProgressProcPtr proc, long refcon);

pascal ComponentResult MovieExportGetAuxiliaryData
    (ComponentInstance ci, Handle dataH,
     OSType *handleType);

pascal ComponentResult MovieExportDoUserDialog
    (ComponentInstance ci, const FSSpec *theFile,
     Handle theData, Boolean *canceled);
```

Pascal Summary

Constants

CONST

{component type values}

```
MovieImportType = 'eat '      {movie data import}
MovieExportType = 'spit'     {movie data export}
```

{componentFlags values for movie import and movie export components}

```
canMovieImportHandles      = 1;  {can import from handles}
canMovieImportFiles        = 2;  {can import from files}
hasMovieImportUserInterface = 4;  {import has user interface}
canMovieExportHandles      = 8;  {can export to handles}
canMovieExportFiles        = $10; {can export to files}
hasMovieExportUserInterface = $20; {export has user interface}
dontAutoFileMovieImport    = $40; {do not automatically import movie }
                             { files}
```

{flags for MovieImportHandle and MovieImportFile functions}

```
movieImportCreateTrack      = 1;  {create a new track}
movieImportInParallel       = 2;  {paste imported data}
movieImportMustUseTrack     = 4;  {use specified track}
movieImportResultUsedMultipleTracks = 8; {component used several }
                             { tracks}
```

{movie data import components}

```
kMovieImportHandleSelect    = 1;  {import from handle}
kMovieImportFileSelect      = 2;  {import from file}
kMovieImportSetSampleDurationSelect = 3; {set sample duration}
```

Movie Data Exchange Components

```

kMovieImportSetSampleDescriptionSelect = 4;  {set sample description}
kMovieImportSetMediaFileSelect         = 5;  {set media file}
kMovieImportSetDimensionsSelect        = 6;  {set track dimensions}
kMovieImportSetChunkSizeSelect         = 7;  {set chunk size}
kMovieImportSetProgressProcSelect      = 8;  {set progress function}
kMovieImportSetAuxiliaryDataSelect     = 9;  {set additional data}
kMovieImportSetFromScrapSelect         = $A;  {data from scrap}
kMovieImportDoUserDialogSelect         = $B;  {invoke user dialog box}
kMovieImportSetDurationSelect          = $C;  {set paste duration}

{movie data export components}
kMovieExportToHandleSelect             = $80;  {export to handle}
kMovieExportToFileSelect               = $81;  {export to file}
kMovieExportDoUserDialogSelect         = $82;  {invoke user dialog box}
kMovieExportGetAuxiliaryDataSelect     = $83;  {get additional data}
kMovieExportSetProgressProcSelect      = $84;  {set progress function}

```

Data Type

TYPE

```

MovieImportComponent = ComponentInstance;
MovieExportComponent = ComponentInstance;

```

Routines

Importing Movie Data

```

FUNCTION MovieImportHandle (ci: MovieImportComponent; dataH: Handle;
                           theMovie: Movie; targetTrack: Track;
                           VAR usedTrack: Track; atTime: TimeValue;
                           VAR addedDuration: TimeValue;
                           inFlags: LongInt; VAR outFlags: LongInt):
  ComponentResult;

FUNCTION MovieImportFile (ci: MovieImportComponent; theFile: FSSpec;
                          theMovie: Movie; targetTrack: Track;
                          VAR usedTrack: Track; atTime: TimeValue;
                          VAR addedDuration: TimeValue;
                          inFlags: LongInt; VAR outFlags: LongInt):
  ComponentResult;

```

Configuring Movie Data Import Components

```

FUNCTION MovieImportSetDuration
    (ci: MovieImportComponent;
     duration: TimeValue): ComponentResult;

FUNCTION MovieImportSetSampleDuration
    (ci: MovieImportComponent; duration: TimeValue;
     scale: TimeScale): ComponentResult;

FUNCTION MovieImportSetSampleDescription
    (ci: MovieImportComponent;
     desc: SampleDescriptionHandle;
     mediaType: OSType): ComponentResult;

FUNCTION MovieImportSetMediaFile
    (ci: MovieImportComponent; alias: AliasHandle):
    ComponentResult;

FUNCTION MovieImportSetDimensions
    (ci: MovieImportComponent;
     width, height: Fixed): ComponentResult;

FUNCTION MovieImportSetChunkSize
    (ci: MovieImportComponent; chunkSize: LongInt):
    ComponentResult;

FUNCTION MovieImportSetProgressProc
    (ci: MovieImportComponent; proc: ProcPtr;
     refCon: LongInt): ComponentResult;

FUNCTION MovieImportSetAuxiliaryData
    (ci: MovieImportComponent; data: Handle;
     handleType: OSType): ComponentResult;

FUNCTION MovieImportSetFromScrap
    (ci: MovieImportComponent; fromScrap: Boolean):
    ComponentResult;

FUNCTION MovieImportDoUserDialog
    (ci: MovieImportComponent; srcFile: FSSpec;
     data: Handle; VAR canceled: Boolean):
    ComponentResult;

```

Exporting Movie Data

```
FUNCTION MovieExportToHandle
    (ci: MovieExportComponent; data: Handle;
     theMovie: Movie; onlyThisTrack: Track;
     startTime: TimeValue; duration: TimeValue):
    ComponentResult;

FUNCTION MovieExportToFile (ci: MovieExportComponent; dstFile: FSSpec;
    theMovie: Movie; onlyThisTrack: Track;
    startTime: TimeValue; duration: TimeValue):
    ComponentResult;
```

Configuring Movie Data Export Components

```
FUNCTION MovieExportSetProgressProc
    (ci: MovieExportComponent; proc: ProcPtr;
     refCon: LongInt): ComponentResult;

FUNCTION MovieExportGetAuxiliaryData
    (ci: MovieExportComponent; dstFile: Handle;
     VAR handleType: OSType): ComponentResult;

FUNCTION MovieExportDoUserDialog
    (ci: MovieExportComponent; dstFile: FSSpec;
     data: Handle; VAR canceled: Boolean):
    ComponentResult;
```

Result Codes

invalidTrack	-2009	Specified track cannot receive imported data
unsupportedAuxiliaryImportData	-2057	Cannot work with specified handle type
badComponentSelector	0x80008002	Function not supported

