QuickTime Import and Export Guide

QuickTime > Import & Export

2006-01-10

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

Introduction to QuickTime Import and Export Guide

This book describes QuickTime's technology for importing and exporting graphics and other data into and out of movies.

QuickTime imports and exports data between movies and still images using graphic importer and exporter components and movie data exchange components:

- Graphic importer and exporter components open, display, and save graphic images stored using various file formats and compression algorithms. Apple-provided components read many common compressed image types. Support for cross-platform and web-derived images is provided, including the ability to query for capability by MIME type and by examination (a component can be asked to examine a file of unknown type, to see if it can decompress the file).
- Movie data exchange components allow you to import data from non-movie sources into QuickTime movies, and to export data to non-movie formats. For example, you can import a CD audio track into a QuickTime movie, or save a QuickTime movie's sound track as an AIFF file, using data exchange components. Applications programmers can use the services of data exchange components indirectly, through high-level calls to the Movie Toolbox. Movie data exchange components can also be controlled directly from applications.

Image importers and exporters manage the import and export of graphic images, such as JPEG, TIFF, Photoshop, and PNG. Movie data exchange components support the import and export of other multimedia formats, such as AIFF, WAVE, AVI, MPEG-1, MIDI, MPEG-4, 3GPP, MP3, MPEG-2, H.263, H.264, and OpenDML. QuickTime can open any format file for which it has an importer and create any for which it has an exporter.

Note: This book replaces three previously separate Apple documents: "Movie Import/Export (Data Exchange Components)," "Graphics Importers," and "Graphics Exporters."

Applications make direct calls to graphic importer components, so this document will be of interest to most QuickTime developers who work with still images or the web. To obtain the services of a graphics importer component, applications normally use the `GetGraphicsImporterForFile` or `GetGraphicsImporterForDataRef` functions of the Image Compression Manager.

If your application needs to import or export data between movies and other data types, you should read the sections "Movie Data Exchange Components" (page 47), and "Using Movie Data Exchange Components" (page 81). If you plan to control data exchange components directly from within your application, or if you are creating a new movie data exchange component, you will need to read all of the material in this document.

If you need to create a new graphics importer component, refer to this document to implement a component that supports the required interface functions.

Organization of This Document

This document is divided into seven chapters:

- "About Graphics Importer and Exporter Components" (page 13) describes what QuickTime graphic importer and exporter components do and shows how to use them from within an application.
- "Graphics Importer Components" (page 19) tells you how applications can use graphics importer components.
- "Graphics Exporter Components" (page 33) describes the general features of graphics exporter components.
- "Graphics Exporter Component Functions By Task" (page 41) describes the graphics exporter functions grouped by task and category.
- "Movie Data Exchange Components" (page 47) describes what movie data exchange components are and shows how they work.
- "Creating a Movie Data Exchange Component" (page 65) describes how to create a movie data exchange component. Sample import and export components are provided as a programming aid.
- "Using Movie Data Exchange Components" (page 81) describes how to use movie exchange components from within your application.

See Also

Sample code is available at:

http://developer.apple.com/samplecode/ImproveYourImage/index.html

The sample code demonstrates the usage of QuickTime graphics importers and exporters, and includes twelve separate examples, each of which can be chosen from the Examples menu. These show how to use graphics importers to

- Simply draw a still image.
- Import, draw, scale and rotate a still image.
- Demonstrate compositing with alpha graphics modes and images containing alpha channels.
- Retrieve metadata from image files.
- Display multiple layers stored in a Photoshop file.
- Import an image from a URL data reference.
- Add a filter to an imported image, draw it with the filter, and export a new image with the filter applied.

The following Apple books cover related aspects of QuickTime programming:

- *QuickTime Overview* gives you the starting information you need to do QuickTime programming.
- *QuickTime Movie Basics* introduces you to some of the basic concepts you need to understand when working with QuickTime movies.

- *QuickTime Movie Creation Guide* describes some of the different ways your application can create a new QuickTime movie.
- *QuickTime Guide for Windows* provides information specific to programming for QuickTime on the Windows platform.
- *QuickTime Media Types and Media Handlers Guide* introduces the idea of QuickTime media handler components and provides details of the video, sound, text, timecode, and tween media handlers.
- *QuickTime Compression and Decompression Guide* introduces you to the QuickTime Image Compression Manager and its associated components, which provide image-compression and image-decompression services to applications and to other QuickTime components.

About Graphics Importer and Exporter Components

This chapter describes what graphic importer and exporter components do and shows how to use them from within an application. A code sample is included.

Graphics Importers

Graphics importer components provide a standard method for applications to open and display still images contained within graphics documents. Graphics importer components allow you to work with any type of supported image data, regardless of the file format or compression used in the file.

You can use the graphics importer component functions to obtain a graphics importer component instance that can read graphics data from a particular file or area of memory. For example, `GetGraphicsImporterForFile` will attempt to locate and open a graphics importer component that can be used to draw the specified file. The component can then be used to find some characteristics of the image such as its dimensions by using `GraphicsImportGetNaturalBounds` and can be drawn by simply calling `GraphicsImportDraw`.

Alpha Channels

Some QuickTime supported file formats, such as TIFF, PNG, and Photoshop, support alpha channels; some, such as JPEG and TGA, do not. However, even if a file format supports alpha channels, not every file will contain one.

You can use graphics importers to find out whether a file has an alpha channel by calling `GraphicsImportGetImageDescription` and looking at the depth field in the image description. By convention, a depth of 24 means RGB without alpha, whereas 32 means ARGB.

QuickTime will not synthesize an opaque alpha channel for an image that does not have one; if the depth is 24, the first byte of each pixel will probably be 0. Developers working with alpha channels should only depend on these values if the depth is 32.

All depths of PNG files can have transparency. For example, a 4-bit indexed-color PNG file can have an opaqueness value associated with each color index. QuickTime translates these files to 32-bit ARGB in order to preserve the transparency information.

Multiple Images

Graphics importers also support image formats containing multiple images in a single file. For example, TIFF files can support multiple images, Photoshop files can contain multiple layers, and FlashPix files can contain multiple resolutions. You can use `GraphicsImportGetImageCount` to find out how many images are in a file, and `GraphicsImportSetImageIndex` to select a particular image.

Graphics Exporters

Graphics exporter components provide a standard interface for exporting graphics to image files in a variety of formats. QuickTime selects a graphic exporter component based on the desired output format, such as GIF or PNG.

The image input for an export operation can come from a QuickDraw picture, a `GWorld` or `PixMap`, a QuickTime graphics importer component instance, or a segment of compressed data described by a QuickTime image description. In the last case, the compressed data may be accessed via a pointer, handle, file or other data reference. The ouput image for an export operation can be stored in a handle, file, or other data reference. Different file formats support a wide range of configurable features, such as depth, resolution, and compression quality.

To use a graphics exporter, you must first open a graphics exporter component instance, specify the source of the input image, its destination, set whatever parameters you want, and then actually do the export.

For example, if you want to write a `GWorld` out to a PNG image file you would would use `OpenADefaultComponent` to open an instance of the `kQTFileTypePNG` graphics exporter component, `GraphicsExportSetInputGWorld` to associate the input `GWorld` with the export operation, `GraphicsExportSetOutputFile` to associate the output `FSSpec` with the export operation and `GraphicsExportDoExport` to perform the export.

Once you're finished with component instances you can simply close them using `CloseComponent`.

Using QuickTime to Export a Picture as an Image File

Graphics importer components provide a simple, flexible and extensible way for you to draw pictures that are stored in a wide variety of image file formats. They also turn out to be the easiest way to export pictures to several file formats.

Typically, you create a graphics importer instance with the `GetGraphicsImporterForFile` call or one of its relatives. Afterwards, you can call `GraphicsImportExportImageFile` to export that image to a new file:

```
GetGraphicsImporterForFile(
                &theExistingFile,
                &gi);

GraphicsImportExportImageFile(
                gi,
                kQTFileTypeJPEG,        // or one of several other file types
                0,                      // ie, the default creator for this type
                &theNewFile,
                theScriptCode);
```

The file type determines the data format for the exported file. For example, if you pass `kQTFileTypeJPEG`, a JPEG file will be created.

If the picture you want to export isn't already in a file, it doesn't make sense to call `GetGraphicsImporterForFile`. Instead, you can create an instance of the "picture file" graphics importer yourself and give it a handle that contains a "source picture file."

```
OpenADefaultComponent(
                GraphicsImporterComponentType,
                kQTFileTypePicture,
                &gi);

GraphicsImportSetDataHandle(
                gi,
                thePictureFileHandle);

GraphicsImportExportImageFile(
                gi,
                kQTFileTypeJPEG,
                0,
                &theNewFile,
                theScriptCode);
```

Picture files are just like picture handles, except that they have an extra 512-byte header at the front. (Any data in this header is usually ignored, but it must be there for historical reasons.) The graphics importer for picture files expects and skips this 512-byte header, so you must insert it.

```
PicHandle thePicture = /* create a picture */;
Handle thePictureFileHandle;

thePictureFileHandle = NewHandleClear(512);
HandAndHand((Handle)thePicture,thePictureFileHandle);
```

The following function exports a picture to a file. The new file is described by its type, creator, specification and script code. If you don't know what script code to use, it's usually safe to use `smSystemScript`.

```
#include <OSUtils.h>
#include <ImageCompression.h>
#include <QuickTimeComponents.h>

OSErr ExportPicHandleToFile(
    PicHandle thePicture,
    OSType filetype,
    OSType filecreator,
    FSSpec *filespec,
    ScriptCode filescriptcode)
{
    Handle h;
    OSErr err;
    GraphicsImportComponent gi = 0;

    // Convert the picture handle into a PICT file (still in a handle)
    // by adding a 512-byte header to the start.
    h = NewHandleClear(512);
    err = MemError();
    if(err) goto bail;
    err = HandAndHand((Handle)thePicture,h);

    err = OpenADefaultComponent(
                GraphicsImporterComponentType,
                kQTFileTypePicture,
                &gi);
    if(err) goto bail;

    err = GraphicsImportSetDataHandle(gi, h);
```

```
    if(err) goto bail;

    err = GraphicsImportExportImageFile(
                gi,
                filetype,
                filecreator,
                filespec,
                filescriptcode);
    if(err) goto bail;

bail:
    if(gi) CloseComponent(gi);
    if(h) DisposeHandle(h);
    return err;
}
```

Your application can find out more information about available export formats by calling the `GraphicsImportGetExportImageTypeList` function. This function returns an atom container (which your application must dispose of) containing several `kGraphicsExportGroup` atoms. Each of these represents one export format, and has child atoms which indicate the file type (`kGraphicsExportFileType`), human-readable format name (`kGraphicsExportDescription`), file extension (`kGraphicsExportExtension`) and optionally MIME type (`kGraphicsExportMIMEType`).

The code fragment in Listing 1-1 shows how you can find out the available export file types. Error-handling code has been removed.

Listing 1-1 Finding the available export file types

```
QTAtomContainer theExportInfo = nil;
short theNumberOfExportTypes, i;

GraphicsImportGetExportImageTypeList(
                gi,
                &theExportInfo);

theNumberOfExportTypes = QTCountChildrenOfType(
                theExportInfo,
                kParentAtomIsContainer,
                kGraphicsExportGroup);        // 'expo'

for(i = 1; i <= theNumberOfExportTypes; i++) {
    QTAtom groupAtom, fileTypeAtom;
    OSType fileType;

    groupAtom = QTFindChildByIndex(
                    theExportInfo,
                    kParentAtomIsContainer,
                    kGraphicsExportGroup,
                    i,
                    nil);

    fileTypeAtom = QTFindChildByIndex(
                    theExportInfo,
                    groupAtom,
                    kGraphicsExportFileType,  // 'ftyp'
                    1,
                    nil);
```

```
    QTCopyAtomDataToPtr(
                    theExportInfo,
                    fileTypeAtom,
                    false,
                    sizeof(fileType),
                    &fileType,
                    nil);

    fileType = EndianU32_BtoN(fileType);       // data in QT atoms
                                               // is always big-endian
}
QTDisposeAtomContainer(theExportInfo);
```

Graphics Importer Components

This chapter describes the functions you can use to obtain the services of graphics importer components and use them to draw and manipulate image files.

Displaying Still Images

Graphics importer components provide a standard method for applications to open and display still images contained within graphics documents. Graphics importer components allow you to work with any type of image data, regardless of the file format or compression used in the document.

You specify the document that contains the image, and the destination rectangle the image should be drawn into, and QuickTime handles the rest. More complex interactions are also supported.

To draw an image file, use the function shown in Listing 2-1.

Listing 2-1 The basic function used to draw an image file

```
void drawFile(const FSSpec *fss, const Rect *boundsRect)
    {
        GraphicsImportComponent gi;
        GetGraphicsImporterForFile(fss, &gi);
        GraphicsImportSetBoundsRect(gi, boundsRect);
        GraphicsImportDraw(gi);
        CloseComponent(gi);
    }
```

The same code can be used to display any image, regardless of the file format.

Image Formats with Multiple Images in a Single File

QuickTime includes support for image formats which can have multiple images in a single file. You can use `GraphicsImportGetImageCount` to find out how many images are in a file, and `GraphicsImportSetImageIndex` to select a particular image. Of the image formats supported by QuickTime, TIFF files can support multiple images, Photoshop files can contain multiple layers and FlashPix files can contain multiple resolutions.

Note that individual images in a file may have different characteristics (width and height, depth, and so on).

Use the following functions to deal with image counts, getting and setting image indexes:

- `GraphicsImportGetImageCount`
- `GraphicsImportSetImageIndex`
- `GraphicsImportGetImageIndex`

Supporting 64-bit File Sizes and Offsets

QuickTime supports 64-bit file sizes and offsets. Four new functions have been added to the graphics importer API. Each is a 64-bit analog of an original 32-bit function. The base graphics importer's implementation of some 32-bit functions have been modified to call the 64-bit version. (They may return `fileOffsetTooBigErr` in the event that a 64-bit value cannot be converted to a 32-bit value.) Applications and format-specific importers can call either version of each function.

Use the following functions to deal with 64-bit file sizes and offsets:

- `GraphicsImportGetDataOffsetAndSize64`
- `GraphicsImportReadData64`
- `GraphicsImportSetDataReferenceOffsetAndLimit64`
- `GraphicsImportGetDataReferenceOffsetAndLimit64`

Retrieving Default Settings

Some file formats, most notably FlashPix, can store a default matrix, clipping region, graphics mode and source rect in the image file. In order to display the image correctly, these settings should be used.

Use the following functions to deal with retrieving default settings:

- `GraphicsImportGetDefaultMatrix`
- `GraphicsImportGetDefaultClip`
- `GraphicsImportGetDefaultGraphicsMode`
- `GraphicsImportGetDefaultSourceRect`

Default settings from FlashPix files are not used automatically.

Getting ColorSyncProfiles

QuickTime includes support for extracting embedded ColorSync profiles from some image formats. Of the image formats supported by QuickTime, these are GIF, JPEG, PNG, QuickDraw Picture, QuickTime Image, and TIFF. Image files containing ColorSync profiles describe their own colorspaces in a self-contained manner.

Since it was introduced in QuickTime, the PNG graphics importer has performed built-in gamma correction. QuickTime includes a flag that can you can set using the `GraphicsImportSetFlags` function to turn this off. This is useful for applications that want to use some other mechanism for colorspace correction, such as ColorSync.

Use the following functions to deal with getting ColorSync profiles, and getting and setting import flags:

- `GraphicsImportGetColorSyncProfile`
- `GraphicsImportSetFlags`
- `GraphicsImportGetFlags`

Getting/Setting the Destination Rectangle

QuickTime includes two calls that let applications access a graphics importer's destination rectangle.

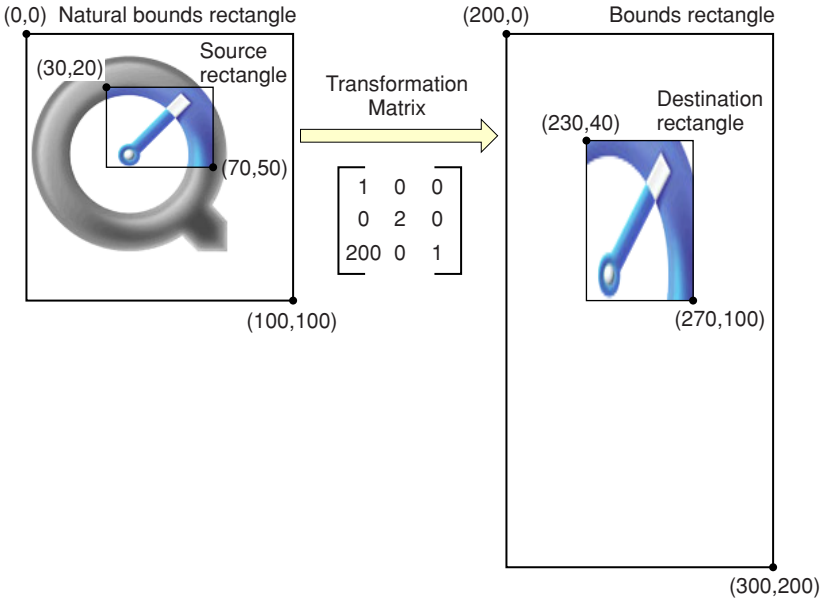
Use the following functions to deal with accessing a destination rectangle:

- `GraphicsImportSetDestRect`
- `GraphicsImportGetDestRect`

Figure 2-1 shows an example with the following four rectangles: natural bounds, source rectangle, bounds rectangle, and a destination rectangle. You can access the destination rectangle in QuickTime with these calls.

In Figure 2-1, the matrix scales by 200% vertically and translates 200 pixels to the right.

Figure 2-1 Matrix scaling and translation



QuickTime Image File Format

QuickTime Image files are intended to provide the most useful container for QuickTime compressed still images. The format uses the same atom-based structure as a QuickTime movie.

Most still image file formats define both how images should be stored and compressed. However, two of the file formats supported by QuickTime are container formats, which describe storage mechanisms independent of compression. These formats are QuickDraw Picture (PICT) files and QuickTime Image (QTIF) files.

QuickTime has permitted compressed image data to be included in QuickDraw pictures since QuickTime was first introduced. However, the technical challenges of parsing, interpreting and spooling picture files can make them a discouraging choice for applications which are primarily interested in accessing the compressed data inside.

The QuickTime Image file format provides a much simpler container for QuickTime compressed still images. The format uses the same atom-based structure as a QuickTime movie. Because the QuickTime Image file is a single fork format, it works well in cross-platform applications. On Mac OS systems, QuickTime Image files are identified by the file type `'qtif'`. On other platforms, Apple recommends that you use the filename extension .QTIF to identify QuickTime Image files.

Atom Types in QuickTime Image Files

There are two defined atom types: `'idsc'`, which contains an image description, and `'idat'`, which contains the image data. They are illustrated in Figure 2-2. For a JPEG image, the image description atom contains a QuickTime image description describing the JPEG image's size, resolution, depth, and so on, and the image data atom contains the actual JPEG compressed data, as shown in Table 2-1.

In QuickTime, there is an optional atom type, `'iicc'`, which can store a ColorSync profile.

Figure 2-2 An 'idsc' atom followed by an 'idat' atom

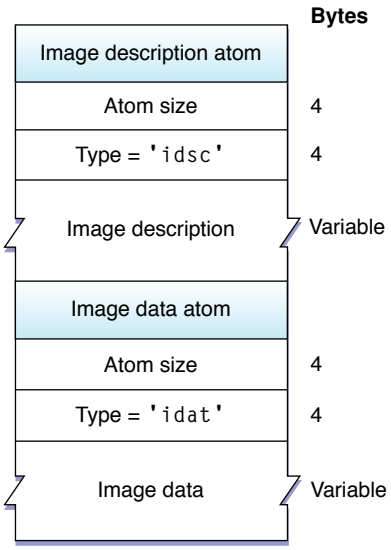


Table 2-1 A QuickTime Image file containing JPEG compressed data

Data	Description
0000005E	Atom size, 94 bytes
69647363	Atom type, `'idsc'`
00000056	Image description size, 86 bytes
6A706567	Compressor identifier, `'jpeg'`

Data	Description
00000000	Reserved, set to zero
0000	Reserved, set to zero
0000	Reserved, set to zero
00000000	Major and minor version of this data, zero if not applicable
6170706C	Vendor who compressed this data, `'appl'`
00000000	Temporal quality, zero (no temporal compression)
00000200	Spatial quality, `codecNormalQuality`
0140	Image width, 320
00F0	Image height, 240
00480000	Horizontal resolution, 72 dpi
00480000	Vertical resolution, 72 dpi
00003C57	Data size, 15447 bytes (use zero if unknown)
0001	Frame count, 1
0C 50 68 6F 74 6F 20 2D 20 4A 50 45 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Compressor name, "Photo - JPEG" (32 byte Pascal string)
0018	Image bit depth, 24
FFFF	Color look-up table ID, -1 (none)
00003C5F	Atom size, 15455 bytes
69646174	Atom type, `'idat'`
FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48 ...	JPEG compressed data

A QuickTime image file can also contain other atoms.

The exact order and size of atoms is not guaranteed to match the example in Table 2-1. Applications reading QuickTime image files should always use the atom size to traverse the file and ignore atoms of unrecognized types.

Like QuickTime movie files, QuickTime Image files are big-endian. However, image data is stored in the same byte order as usually specified by the particular compression format.

Obtaining Graphics Import Components

You can use the `GetGraphicsImporterForFile` function to open a suitable graphics import component for a file. If you have a data reference instead of a file, you can use `GetGraphicsImporterForDataRef` instead.

When the Image Compression Manager's `GetGraphicsImporterForFile` function searches for a graphics import component, it tries, in order:

- matching the Mac OS file type
- matching the file name suffix
- matching the MIME type and MIME suggested file name suffix
- asking individual graphics importer components if the file's contents match their format. It only does this for those graphics importers which support the `GraphicsImporterValidate` call.

The last `stage` can be time-consuming, since it involves opening many components in turn. If you want to skip it, call `GetGraphicsImporterForFileWithFlags` (or `GetGraphicsImporterForDataRefWithFlags`) and pass the `kDontUseValidateToFindGraphicsImporter` flag.

If you include `kQTFileTypeQuickTimeImage` (`'qtif'`) in the list of types passed to `StandardGetFilePreview`, all files that can be opened with graphics importers are included in the file list. The slow validate approach is not used in this case.

When you are done with the graphics importer instance, you call `CloseComponent`.

If you expect to draw the same image more than once, you can improve performance by keeping the graphics importer component open, rather than creating and disposing of it each time.

Once you have a graphics import component for your file or data reference, you can query it to determine the properties of the file, configure drawing parameters, and draw or export the file. The next two sections explain how to do this.

Determining the Properties of the Image File

If you want to know the dimensions of the image, call the importer's `GraphicsImportGetNaturalBounds` function. If you want to know other information that is represented in the image description, such as its depth or color table, call `GraphicsImportGetImageDescription`. If you want to extract meta-data from the image file (such as textual comments, like copyright information), call `GraphicsImportGetMetaData`. If you want to know information about the file format, such as the MIME types and MIME suggested file name suffixes, call `GraphicsImportGetMIMETypeList`.

Some image file formats can contain transparent regions, and hence may leave some pixels in their rectangular range unmodified even when drawing with a copying transfer mode and an identity matrix. If you would like to know whether a graphics importer's format supports such transparent regions, call `GraphicsImportDoesDrawAllPixels`. If a graphics importer doesn't support the `GraphicsImportDoesDrawAllPixels` call, you can assume it will draw all pixels.

Drawing and Converting Image Files

Before drawing, you may wish to set various parameters. Among those you can configure are the

- source rectangle
- transformation matrix
- clipping region
- graphics transfer mode
- drawing quality
- destination graphics port and device

These parameters are explained in detail in Table 2-2.

Table 2-2 Drawing parameters you may configure

Parameters	Description
source rectangle	Used to select a rectangular portion of the compressed image.
transformation matrix	Used to shift, scale, rotate, and apply perspective to the source portion of the image. (Set this with `SetMatrix`, `SetBoundsRect`, or `SetDestRect`.)
clipping region	Used to restrict the area to be drawn in the destination space.
graphics transfer mode	Used to define how source pixels modify destination pixels.
drawing quality	Used to specify quality vs. time tradeoffs; For example, if you're drawing a JPEG image to an 8-bit color screen, the drawing quality determines whether a slower or faster dither will be used.
destination graphics port and device	Defines the graphics environment for drawing.

Once you have set the drawing parameters, you can call `Draw` to draw the image.

You can also call `GraphicsImportGetAsPicture` to get the image in the form of a QuickDraw picture handle, `GraphicsImportSaveAsPicture` to save it in a PICT file, or `GraphicsImportSaveAsQuickTimeImage` to save it in a QuickTime Image file. To export it to other image file formats, you can use `GraphicsImportExportImageFile`, or `GraphicsImportDoExportImageFileDialog` to present a standard Export As.. dialog box.

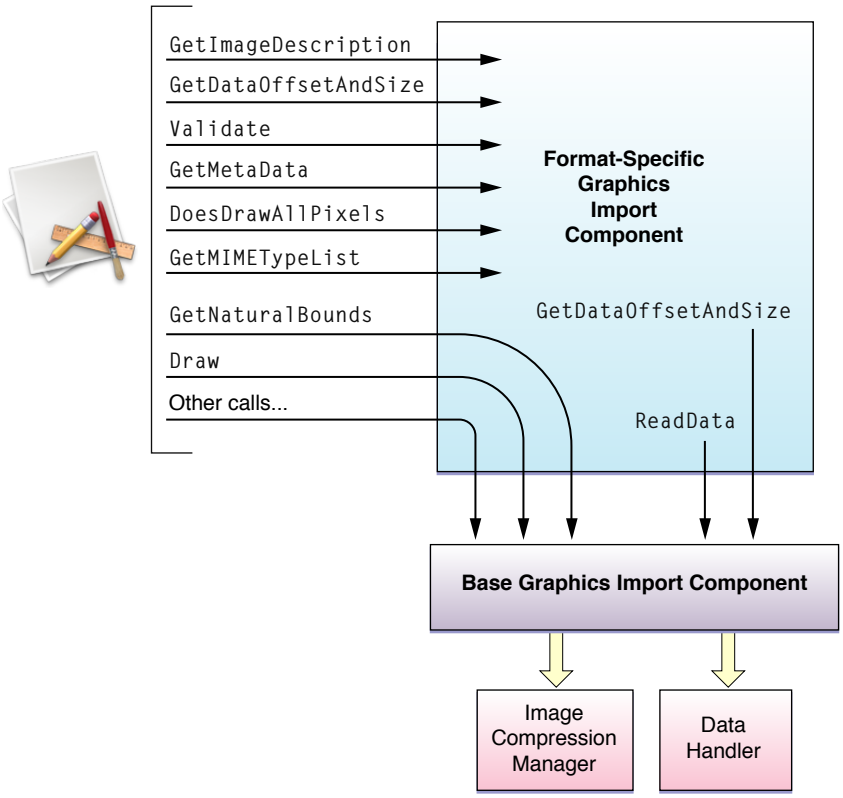
When you are done with a graphics import component, you call `CloseComponent`.

Writing Graphics Import Components

This section describes how graphics import components work and briefly discusses how to write a component. If you're interesting in writing a component, you should read this section.

Format-specific graphics import components, such as the importers for JPEG, PNG, TIFF etc., are simple components. When a format-specific graphics importer is opened, it opens and targets an instance of the base importer. Subsequently, it delegates most of its calls to the base importer instance, as shown in Figure 2-3.

Figure 2-3 Delegating calls to the base importer



The base importer communicates with data handler components to access image file data, and with the Image Compression Manager to arrange for image rendering. The only service a format-specific importer must provide is the `GraphicsImportGetImageDescription` call, which examines an image file and constructs an image description for it. The base importer uses this image description to respond to other calls such as `GraphicsImportGetNaturalBounds` and `GraphicsImportDraw`. (In the case of `GraphicsImportDraw`, the image description is passed to the Image Compression Manager, so the `cType` field must identify a codec that will be able to draw the image. If a graphics importer needs to pass extra information that the codec will need at `PreDecompress` time, it can pass it in an image description extension.)

Sometimes, the data to be passed to the image decompressor is only a portion of the file. In these cases, the format-specific importer should implement the `GraphicsImportGetDataOffsetAndSize` function to indicate the byte range to send to the image decompressor. It is often useful for this call to first call the generic importer's implementation to find out the size of the input data stream.

In QuickTime, there is a 64-bit analog of this function, called `GraphicsImportGetDataOffsetAndSize64`. In order to provide compatibility with old graphics importers, the generic importer's implementation of this function calls the 32-bit version. Format-specific importers may implement both the 32-bit and 64-bit versions if it makes sense for their file formats.

Graphics import components may override other calls, such as `GraphicsImportGetMetaData`, which extracts supplemental information from an image file, and `GraphicsImportGetMIMETypeList`, which provides information about the format.

Another optional call is `GraphicsImportValidate`, which attempts to ascertain quickly whether a file matches the importer's format. This is especially useful for formats which start with identifying codes or "magic numbers" (such as PNG and TIFF) in situations where image files do not have correct file types or suffixes. In situations like this, the Image Compression Manager may ask many graphics importers in turn to validate until it finds one that accepts the file, so it is important that `GraphicsImportValidate` calls not be too slow. Format-specific importers that implement the `GraphicsImportValidate` call should have the `canMovieImportValidateFile` bit set in their component flags.

Graphics importers supporting image formats which can have transparent regions should implement the `GraphicsImportDoesDrawAllPixels` call so as to warn applications that they may need to erase the destination area before drawing.

Registering Graphics Import Components

Graphics import components have component type `'grip'`. The interpretation of the subtype depends on the `movieImportSubTypeIsFileExtension` component flag. If this flag is clear, the subtype is a Macintosh file type. If this flag is set, the subtype is a file name suffix; it should be in uppercase and followed by space characters to pad it out to four characters. For instance, the file name suffix .png would be represented by the subtype `'PNG '`.

It is often useful to register graphics import components multiple times, so that both the file type and file name suffix may be matched. An efficient way to do this is to register the second and subsequent components as component aliases to the first.

Graphics import components that use the base importer's `Draw` method should set the `graphicsImporterUsesImageDecompressor` flag in their component flags.

Getting Image Characteristics

These functions are called by applications to obtain information about images:

- `GraphicsImportGetNaturalBounds`
- `GraphicsImportGetImageDescription`
- `GraphicsImportGetDataOffsetAndSize`
- `GraphicsImportValidate`
- `GraphicsImportGetMetaData`
- `GraphicsImportDoesDrawAllPixels`

Format-specific graphics importers always implement `GraphicsImportGetImageDescription` and may optionally implement the remaining functions listed above.

Setting Drawing Parameters

The functions listed below allow you to specify various parameters for drawing operations, such as clipping, scaling, graphics mode, and decompression quality. All of these functions are based on corresponding routines in the Image Compression Manager for working with image decompression sequences:

- `GraphicsImportSetBoundsRect`
- `GraphicsImportGetBoundsRect`
- `GraphicsImportSetMatrix`
- `GraphicsImportGetMatrix`
- `GraphicsImportSetClip`
- `GraphicsImportGetClip`
- `GraphicsImportSetGraphicsMode`
- `GraphicsImportGetGraphicsMode`
- `GraphicsImportSetQuality`
- `GraphicsImportGetQuality`
- `GraphicsImportSetSourceRect`
- `GraphicsImportGetSourceRect`
- `GraphicsImportSetProgressProc`
- `GraphicsImportGetProgressProc`

Drawing Images

These functions are used to draw images:

- `GraphicsImportSetGWorld`
- `GraphicsImportGetGWorld`
- `GraphicsImportDraw`

Saving Image Files

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use the `GraphicsImportDraw` function.

- `GraphicsImportSaveAsPicture`
- `GraphicsImportSaveAsQuickTimeImageFile`

- GraphicsImportGetAsPicture
- GraphicsImportExportImageFile
- GraphicsImportGetExportImageTypeList
- GraphicsImportDoExportImageFileDialog
- GraphicsImportGetExportSettingsAsAtomContainer
- GraphicsImportSetExportSettingsFromAtomContainer

Getting MIME Types

Your graphics import component can support MIME types that correspond to graphics formats it supports. To make a list of these MIME types available to applications or other software, it must implement this function:

- GraphicsImportGetMIMETypeList

Specifying the Data Source

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use the graphics importer component functions described in this section, rather than directly calling a data handler. These functions allow the data source to be a file, a handle, or a QuickTime data reference.

You do not need to call the functions in this section if you use one of the GetGraphicsImporter functions. The GetGraphicsImporter functions automatically set the graphics importer component's data source. You only need to use these functions if you open the graphics importer component directly.

- GraphicsImportSetDataFile
- GraphicsImportGetDataFile
- GraphicsImportSetDataHandle
- GraphicsImportGetDataHandle
- GraphicsImportSetDataReference
- GraphicsImportGetDataReference
- GraphicsImportSetDataReferenceOffsetAndLimit
- GraphicsImportGetDataReferenceOffsetAndLimit

Retrieving Image Data

This function is used by format-specific graphics import components to read data from the data source; it is implemented by the base graphics importer:

- `GraphicsImportReadData`

Graphics Importer Flags for Gamma Correction

You can set the `kGraphicsImporterDontDoGammaCorrection` flag when you want to tell the Graphics Importer not to perform gamma correction

```
enum {
    kGraphicsImporterDontDoGammaCorrection = 1L
};
```

Term	Definition
`kGraphicsImporterDontDoGammaCorrection`	Specifies not to perform a gamma correction.

Image Description Atoms in QuickTime Image Files

These atoms may appear in QuickTime image files:

```
enum {
    quickTimeImageFileImageDescriptionAtom = FOUR_CHAR_CODE('idsc'),
    quickTimeImageFileImageDataAtom = FOUR_CHAR_CODE('idat'),
    quickTimeImageFileMetaDataAtom = FOUR_CHAR_CODE('meta'),
};
```

ColorSync Atoms in QuickTime Image Files

QuickTime includes a ColorSync atom type, which can be used to store ColorSync profile information:

```
enum {
    quickTimeImageFileColorSyncProfileAtom = FOUR_CHAR_CODE('iicc')
};
```

Graphics Importer Component Type

Graphics importer components have this component type:

```
enum {
    GraphicsImporterComponentType = 'grip'
};
```

MIME Type List

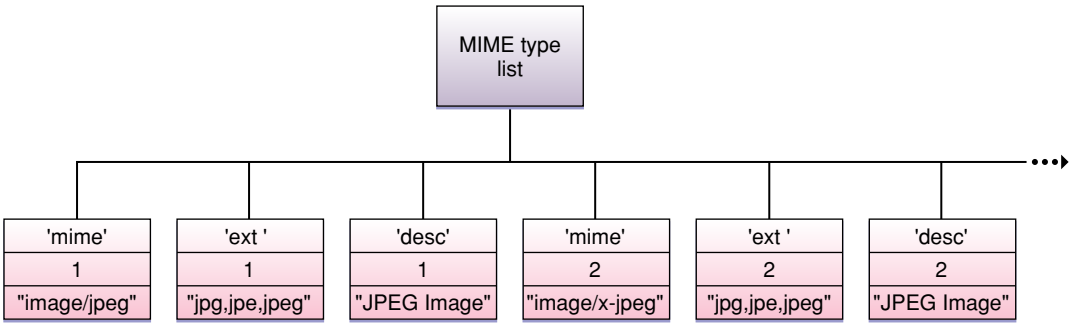
The `GraphicsImportGetMIMETypeList` function returns a list of the MIME types supported by a graphics importer component. This list is contained in the QT atom container described in this section.

At the top level of the atom container are three atoms for each supported MIME type. The atoms whose IDs are 1 describe the first supported MIME type, the atoms whose IDs are 2 describe the second supported MIME type, and so on. Note that the IDs have to be consecutive.

- An atom of type `kMimeInfoMimeTypeTag` contains a string that identifies the MIME type, such as `image/jpeg` or `image/x-jpeg`.
- The atom of type `kMimeInfoFileExtensionTag` contains a string that specifies likely file extensions for files of this MIME type, such as `jpg`, `jpe`, and `jpeg`. If there is more than one extension, the extensions are separated by commas.
- The atom of type `kMimeInfoDescriptionTag` contains a string describing the MIME type for end users, such as `JPEG Image`. These are neither Pascal nor C strings; they are just ASCII characters.

Figure 2-4 illustrates a MIME type list.

Figure 2-4 A MIME type list



Graphics Exporter Components

This chapter describes the general features of graphics exporter components.

Applications make direct calls to graphic exporter components, so this chapter will be of interest to most QuickTime developers who work with still images or the web. Much of the conceptual information found in "Graphics Importer Components" (page 19) also applies to exporters. You should be familiar with that chapter before reading this chapter.

Applications programmers should also be familiar with the Component Manager before working with graphics exporter components.

If you would like to create a new graphics exporter component, refer to this chapter to implement a component that supports the required interface functions.

Exporting Graphics

Graphics exporter components provide a standard interface for exporting graphics to image files in a variety of formats. QuickTime selects a graphic exporter component based on the desired output format, such as GIF or PNG. Different image formats support a wide range of configurable features, such as depth, resolution and compression quality.

QuickTime supports the following export image formats: BMP, JFIF, JPEG, GIF, MacPaint, Photoshop, PNG, QuickDraw PICT, QuickTime Image, Silicon Graphics, Targa, and TIFF. Note that this list is not inclusive; QuickTime supports additional export image formats with each software release.

A graphic image can be exported to a handle, a file, or a data reference.

Third-party developers may also write their own graphics exporters for other image file formats.

Using Graphics Exporter Components

This section describes how to use graphics exporter components. Code samples are included.

To export an image to a file or a data reference using a graphics exporter, you must open a graphics exporter component instance, specify the source of the input image and the destination for the output image file, set the parameters you want, and call `GraphicsExportDoExport`.

You select a particular graphics exporter component based on the desired output format, such as GIF or PNG. If you know which file format you want to write, you can use a call such as `OpenADefaultComponent` to open the appropriate exporter.

The input image can come from a QuickDraw `Picture`, a `GWorld` or `PixMap`, a QuickTime graphics importer component instance, or a segment of compressed data described by a QuickTime image description. In the last case, the compressed data may be accessed via a pointer, handle, file or other data reference.

The output image for an export operation can be specified as a handle, a file, or a data reference.

When you are done with the graphics exporter component instance, call `CloseComponent`.

The example code in Listing 3-1 shows the basic functions you use to export a `GWorld` to an image file. Error-checking code has been removed.

Listing 3-1 The basic functions used to export a GWorld to an image file

```
void writeGWorldToImageFile( GWorldPtr gw, const FSSpec *fileSpec )
{
    GraphicsExportComponent ge = 0;

    OpenADefaultComponent( GraphicsExporterComponentType,
                           kQTFileTypePNG, &ge );
    GraphicsExportSetInputGWorld( ge, gw );
    GraphicsExportSetOutputFile( ge, fileSpec );
    GraphicsExportDoExport( ge, nil );
    CloseComponent( ge );
}
```

Selecting a Graphics Exporter Component

Graphics exporter components have component type `'grex'` and, by convention, component subtype matching the Mac OS file type for the image file format. For example, the graphics exporter for PNG files has component subtype `'PNGf'`.

If you know which file format you want to write, you can use a call such as `OpenADefaultComponent` to open the appropriate exporter.

If you want to display a list of graphics exporter components and let the user select one, you can build a list by using `FindNextComponent` to search for graphics exporter components and calling `GetComponentInfo` on each to find out its name, as shown in Listing 3-2.

Take care not to offer the base exporter as an option. An easy way to do this is to restrict your search to those graphics exporter components that do not have the `graphicsExporterIsBaseExporter` component flag set.

Listing 3-2 Building a list of graphics exporter components

```
void findGraphicsExporterComponents()
{
    ComponentDescription cd;
    Component c = 0;

    cd.componentType = GraphicsExporterComponentType;
    cd.componentSubType = 0;
    cd.componentManufacturer = 0;
    cd.componentFlags = 0;
    cd.componentFlagsMask = graphicsExporterIsBaseExporter;
```

```
    while( ( c = FindNextComponent( c, &cd ) ) != 0 ) {
        // add component c to the list.
    }
}
```

Specifying the Source

The input image for an export operation can come from a QuickDraw Picture, `GWorld` or `PixMap`, or a QuickTime graphics importer instance. To specify such a source, call the respective function:

```
GraphicsExportSetInputPicture
GraphicsExportSetInputGWorld
GraphicsExportSetInputPixMap
GraphicsExportSetInputGraphicsImporter
```

Alternatively, the input image may be a segment of compressed data described by an image description. To specify such a source, call one of these functions:

```
GraphicsExportSetInputDataReference
GraphicsExportSetInputFile
GraphicsExportSetInputHandle
GraphicsExportSetInputPtr
```

In this case, the compressed data may comprise only a segment of the whole data reference. You may limit access to a segment by calling the `GraphicsExportSetInputOffsetAndLimit`.

Specifying the Destination

The output image for an export operation can be written to a handle, file, or other data reference. To specify the destination, call the respective function:

```
GraphicsExportSetOutputDataReference
GraphicsExportSetOutputFile
GraphicsExportSetOutputHandle
```

By default, the graphics exporter's suggested file type and creator will be used for any newly created file. To override the suggested file type or creator, call `GraphicsExportSetOutputFileTypeAndCreator`.

Specifying Export Settings

A variety of `GraphicsExportSet`... and `GraphicsExportGet`... functions are available to set and get various settings. Not every graphics exporter supports all of these.

As usual, if you call an unimplemented component function, it will return `badComponentSelector`. You can find out whether a particular component function is implemented by a component with the `CallComponentCanDo` function.

DontRecompress Flag

Repeated compression of an image with a lossy format (such as JPEG) can cause image degradation.

The `DontRecompress` flag is used to request that the original compressed data not be decompressed and recompressed, but be copied (possibly with modifications) through to the output file. This is not always possible; circumstances may still force a graphics exporter to recompress.

In QuickTime, the JPEG, PICT and QuickTime Image graphics exporters support the `DontRecompress` flag.

Interlace Styles

Some image formats support interlacing, in which image data is rearranged in some manner. A common use for this is in the PNG and GIF formats, which rearrange data so that low-resolution images can be displayed from incomplete data streams.

In QuickTime, the PNG graphics exporter supports the `InterlaceStyle` settings `kQTPNGInterlaceNone` and `kQTPNGInterlaceAdam7`.

MetaData

Some image file formats can contain supplemental data, such as textual copyright information.

In QuickTime, none of the supplied graphics exporters support setting `MetaData`.

Compression Methods

Some image file formats support more than one compression algorithm. In QuickTime, the TIFF graphics exporter supports the `CompressionMethod` settings `kQTTIFFCompression_None` and `kQTTIFFCompression_PackBits`.

Compression Quality

Lossy image compression involves a quality-versus-size tradeoff: the higher the image quality, the greater the compressed data size.

In QuickTime, the JPEG, PICT and QuickTime Image graphics exporters support the `CompressionQuality` setting. The PICT and QuickTime image graphics exporters pass it on as a parameter to a compressor.

Target Data Size

An alternative way to view the quality-versus-size tradeoff is to specify a desired maximum data size and ask for a quality that does not exceed that size.

In QuickTime, the JPEG graphics exporter supports the `TargetDataSize` setting, which is implemented by compressing the image repeatedly, lowering the quality until the target size (or a maximum attempt limit) is reached.

Resolution

Some image file formats can store the image resolution.

In QuickTime, the BMP, JPEG, Photoshop, PNG, QuickTime Image, and TIFF graphics exporters support the `resolution` setting. If you do not set a resolution explicitly, the original image's resolution is used.

Depth

Some image file formats support more than one pixel depth.

In QuickTime, the BMP, JPEG, Photoshop, PNG, PICT, QuickTime Image, Targa and TIFF graphics exporters support the `depth` setting.

Note that the usual QuickDraw conventions for bit depths apply:

- 24 means "Millions of Colors"
- 32 means "Millions+," meaning millions of colors plus an alpha channel
- 40 means 8-bit grayscale.

ColorSyncProfile

Some image file formats support embedded ColorSync profiles. This allows image files to describe their native colorspace in a self-contained manner. In QuickTime, the JPEG, PNG, PICT, QuickTime Image and TIFF graphics exporters support embedded ColorSync profiles.

Settings Dialogs

You can access all of a graphics exporter's settings at once with the `GraphicsExportSetSettingsFromAtomContainer` and `GraphicsExportGetSettingsAsAtomContainer` functions. These functions can also be used to access other settings besides those in the list above.

User Interface

Some graphics exporters can display a dialog to let the user configure format-specific settings. To display such a dialog, call the `GraphicsExportRequestSettings` function.

In QuickTime, the BMP, JPEG, Photoshop, PNG, PICT, QuickTime Image and TIFF graphics exporters support `settings` dialogs.

You can get a user-readable description of a graphics exporter's current settings by calling the `GraphicsExportGetSettingsAsText` function.

File Type and Creator

To find out the suggested file name extension for a graphics exporter's format, call `GraphicsExportGetDefaultFileNameExtension`.

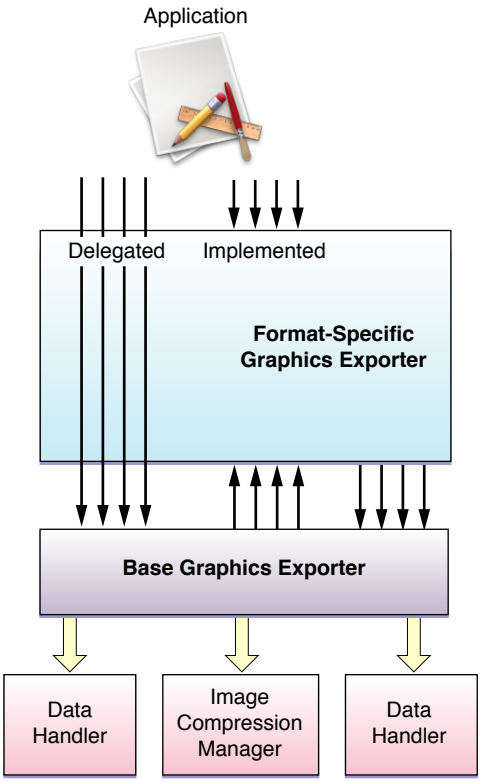
To find out the suggested Mac OS file type and creator, call `GraphicsExportGetDefaultFileTypeAndCreator`. To find out the MIME type, call `GraphicsExportGetMIMETypeList`.

How Graphics Exporters Work

QuickTime provides a base graphics export component which provides abstractions that greatly simplify the work of format-specific graphics exporter components, while offering applications a rich interface.

Format-specific graphics exporters, such as the exporters for JPEG, PNG and TIFF, are relatively simple components. When a format-specific exporter is opened, it opens and targets an instance of the base graphics exporter. Subsequently, it delegates most of its calls to the base exporter instance, as shown in Figure 3-1.

Figure 3-1 Delegating calls to the base graphics exporter



The base exporter communicates with data handler components to write image file data. If necessary, it calls the Image Compression Manager to perform compression operations.

There are three modes that format-specific exporters can operate in:

- Transcode. Data is transferred from one image file to another without decompressing and recompressing. (The data may be modified along the way, if appropriate.)
- Using a Compressor. The graphics exporter provides an atom container identifying which compressor to use, and any settings to be passed to that compressor.
- Standalone Export. The graphics exporter component does all the work itself, without using an image compressor.

How the Base Exporter Chooses a Mode

When an application calls `GraphicsExportDoExport`, the base exporter asks the following question:

- Can QuickTime transcode?

It calls the format-specific exporter's `GraphicsExportCanTranscode` function. If the answer is yes, it calls the exporter's `GraphicsExportDoTranscode` function.

Otherwise,

- Can QuickTime use a compressor?

It calls the format-specific exporter's `GraphicsExportCanUseCompressor` function.

If the answer is yes, it calls the Image Compression Manager to compress using the compressor and parameters specified in the atom container. This is done in the base exporter's `GraphicsExportDoUseCompressor` function. Usually the format-specific exporter does not need to override this function and should simply delegate it. If the format-specific exporter's format is a container format (such as a PICT or QuickTime Image), it can override and delegate this in order to encapsulate the compressed data in the container format.

Otherwise,

- QuickTime must perform a standalone export.

If neither transcoding nor compressing is appropriate, it calls the format-specific exporter's `GraphicsExportDoStandaloneExport` function.

Writing Graphics Export Components

If you're interesting in writing a component, you should check out the sample code at

`http://developer.apple.com/samplecode/ElectricImageComponent/ElectricImageComponent.html`

The sample code demonstrates how to build five QuickTime Components: a graphics importer, graphics exporter, movie importer, movie exporter and image decompressor.

These components all work together to allow QuickTime to use the Electric Image format image files. Sample Electric Image files are included.

Graphics Exporter Component Functions By Task

This chapter describes the constants, data types, and functions in the Graphics Exporter Component.

Constants

The following constants are used to specify graphics exporter characteristics:

```
enum {
    GraphicsExporterComponentType = FOUR_CHAR_CODE('grex'),
    kBaseGraphicsExporterSubType = FOUR_CHAR_CODE('base')
};

enum {
    graphicsExporterIsBaseExporter = 1L << 0,
    graphicsExporterCanTranscode = 1L << 1,
    graphicsExporterUsesImageCompressor = 1L << 2
};
```

Data Types

The graphics exporter component type and subtype are defined as follows:

```
typedef ComponentInstance GraphicsExportComponent;
```

Functions

The graphics export functions listed in the following sections are grouped by task.

Exporting

This function is used to perform export operations.

- `GraphicsExportDoExport`

Internal Routines

These functions are used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call them.

- `GraphicsExportCanTranscode`
- `GraphicsExportDoTranscode`
- `GraphicsExportCanUseCompressor`
- `GraphicsExportDoUseCompressor`
- `GraphicsExportDoStandaloneExport`

Finding Out About Image Formats

These functions return information about the image format supported by a graphics exporter. Format-specific exporters must implement all three of these calls.

- `GraphicsExportGetDefaultFileTypeAndCreator`
- `GraphicsExportGetDefaultFileNameExtension`
- `GraphicsExportGetMIMETypeList`

Obtaining Graphics Exporter Settings

These functions are used for obtaining graphics exporter settings and displaying a settings dialog box.

- `GraphicsExportRequestSettings`
- `GraphicsExportSetSettingsFromAtomContainer`
- `GraphicsExportGetSettingsAsAtomContainer`
- `GraphicsExportGetSettingsAsText`

Accessing Graphics Exporter Settings

Graphics exporters may implement some or none of these functions. To determine whether a particular setting is available, use `CallComponentCanDo`.

- `GraphicsExportSetDontRecompress`
- `GraphicsExportGetDontRecompress`
- `GraphicsExportSetInterlaceStyle`
- `GraphicsExportGetInterlaceStyle`
- `GraphicsExportSetMetaData`
- `GraphicsExportGetMetaData`

- GraphicsExportSetTargetDataSize
- GraphicsExportGetTargetDataSize
- GraphicsExportSetCompressionMethod
- GraphicsExportGetCompressionMethod
- GraphicsExportSetCompressionQuality
- GraphicsExportGetCompressionQuality
- GraphicsExportSetResolution
- GraphicsExportGetResolution
- GraphicsExportSetDepth
- GraphicsExportGetDepth
- GraphicsExportSetColorSyncProfile
- GraphicsExportGetColorSyncProfile

Getting and Setting Progress Procs

These functions are always implemented by the base graphics exporter.

- GraphicsExportSetProgressProc
- GraphicsExportGetProgressProc

Specifying Sources for Input Images

These functions specify the source for input images. You must specify a source before you can call GraphicsExportDoExport.

The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a GWorld or a PixMap. Or it can be a piece of compressed data described by an image description. Compressed data can be in a file, handle, pointer, or other data reference.

The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source.

All of these functions are implemented by the base graphics exporter. Format-specific importers should delegate all of them.

- GraphicsExportSetInputDataReference
- GraphicsExportGetInputDataReference
- GraphicsExportSetInputFile
- GraphicsExportGetInputFile
- GraphicsExportSetInputHandle
- GraphicsExportGetInputHandle

- GraphicsExportSetInputPtr
- GraphicsExportGetInputPtr
- GraphicsExportSetInputGraphicsImporter
- GraphicsExportGetInputGraphicsImporter
- GraphicsExportSetInputPicture
- GraphicsExportGetInputPicture
- GraphicsExportSetInputGWorld
- GraphicsExportGetInputGWorld
- GraphicsExportSetInputPixmap
- GraphicsExportGetInputPixmap

Restricting the Range of an Input Image's Source

These functions are only applicable when the input is a data reference, file, handle or pointer.

- GraphicsExportSetInputOffsetAndLimit
- GraphicsExportGetInputOffsetAndLimit

Reading Input Data

These functions are used by format-specific graphics exporters when transcoding. Applications will not normally need to call these functions.

- GraphicsExportMayExporterReadInputData
- GraphicsExportGetInputDataSize
- GraphicsExportReadInputData

Accessing the Input Image

These functions are used by format-specific graphics exporters. When doing a standalone export, an exporter will typically call GraphicsExportGetInputImageDescription or GraphicsExportGetInputImageDimensions and GraphicsExportGetInputImageDepth to determine the image's bounds and depth, and then allocate an offscreen GWorld and call GraphicsExportDrawInputImage to draw portions of the image into the GWorld.

- GraphicsExportGetInputImageDescription
- GraphicsExportGetInputImageDimensions
- GraphicsExportGetInputImageDepth
- GraphicsExportDrawInputImage

Destinations for Output Images

These functions are used to specify destinations for output images.

- `GraphicsExportSetOutputDataReference`
- `GraphicsExportGetOutputDataReference`
- `GraphicsExportSetOutputFile`
- `GraphicsExportGetOutputFile`
- `GraphicsExportSetOutputHandle`
- `GraphicsExportGetOutputHandle`
- `GraphicsExportSetOutputOffsetAndMaxSize`
- `GraphicsExportGetOutputOffsetAndMaxSize`
- `GraphicsExportSetOutputFileTypeAndCreator`
- `GraphicsExportGetOutputFileTypeAndCreator`

Writing Output Data

These functions are used by format-specific graphics exporters to write output data.

- `GraphicsExportWriteOutputData`
- `GraphicsExportSetOutputMark`
- `GraphicsExportGetOutputMark`
- `GraphicsExportReadOutputData`

Movie Data Exchange Components

This chapter provides background information about movie data exchange components. After reading this chapter, 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.

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 5-1 shows the relationship between the Movie Toolbox and movie data import components while Figure 5-2 shows how movie data export components fit into the picture.

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.

Figure 5-1 The Movie Toolbox, movie data import components, and your application

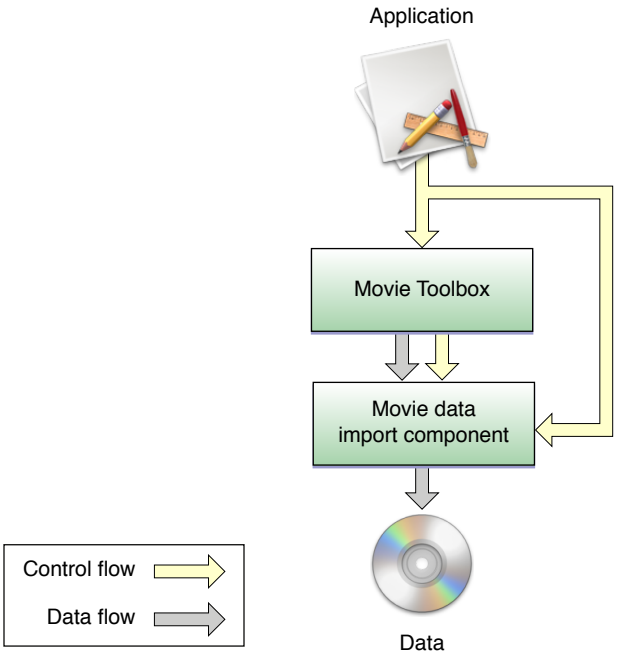
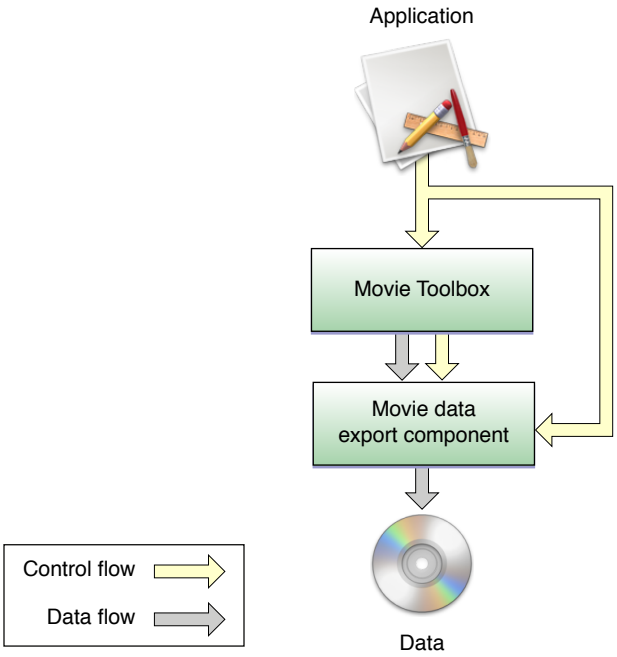


Figure 5-2 The Movie Toolbox, movie data export components, and your application



Saving and Restoring Settings

QuickTime has always provided many support mechanisms for importing media from other formats into QuickTime movies, as well as exporting from QuickTime movies to other media formats. Importing and exporting are handled by movie data exchange components.

You write import and export components to allow a user to perform importing and exporting, respectively. Your component provides a routine that presents a dialog box for the user to change options. For an import component, you need to implement `MovieImportDoUserDialog`; for an export component, you must implement `MovieExportDoUserDialog`. For example, the text import component presents a dialog box with options for setting the font, size, and style of the text media it will add to the movie. The WAVE audio export component presents the standard sound compression dialog box, so that sample rate and sample size can be specified for the generated WAVE file.

QuickTime lets you retrieve the current settings from the still-open import or export component. In addition, you can restore a component's current settings to previously-retrieved settings. The restoration does not involve any user interface. This may be advantageous for application developers who want to provide preferences for the last settings used or want to perform batch importing or exporting, using previously-established settings.

QuickTime makes it possible for your application to retrieve and store the settings of import and export components without having to present the user with a user interface, such as a settings dialog box, to accomplish the task.

Two scenarios illustrate how saved settings can be useful. In the first scenario, an application presents an importer or exporter component's configuration dialog the first time that component is used and then saves the settings so they can be restored without the user having to go fill out the configuration dialog again. In another scenario, an application might use settings to implement preset configurations that the user often wants.

QuickTime enables movie export components to associate resources that hold one or more named presets for that exporter. The dialog accessible through `ConvertMovieToFile` automatically builds a menu of all presets for the currently selected exporter allowing a user to export without having to go through the exporter's custom dialog.

It is also possible to include component resources that serve as named presets to be used with the export component. These resources include the same kind of settings just described. See "Movie Exporter Presets" (page 49).

For information about using the save-and-restore component settings mechanism, refer to the section "Implementing Movie Data Exchange Components" (page 49).

Movie Exporter Presets

The `ConvertMovieToFile` function retrieves preset information and includes an additional popup menu showing presets for the currently selected exporter. Current preset component resources include `'stg#'` and `'sttg'`.

Implementing Movie Data Exchange Components

The following section discusses how you can implement component routines to save and restore component settings available in QuickTime.

Standard Compression Components and Settings

QuickTime includes two settings-related component calls to both the video and sound Standard Compression components: `SCGetSettingsAsAtomContainer` and `SCGetSettingsAsAtomContainer` These may also be useful for implementing movie data exchange components. The `SCGetSettingsAsAtomContainer` routine returns a QT atom container with the current configuration of the particular compression component. `SCSetSettingsFromAtomContainer` resets the compression component's current configuration. Applications that want to save settings for standard compression components should use these calls.

Exporting Text

The text export and import components provide features that make it easier to work with the data in a text track in a QuickTime movie. **Text descriptors** are formatting commands that you can embed within a text file. **Time stamps** describe a text sample's starting time and duration.

The text export and import components make it easier to edit and format text using an external tool, such as a text editor or word processor. When you export text from a text track, you can optionally export text descriptors and time stamps for the text. You can open the text file in a word processor and make changes to the text, style, color, and time stamps. You can then import the edited text to a text track where all the timing, style, color and time stamp information will be present.

When you export text, you control whether text descriptors and time stamps are to be exported by selecting the appropriate options in the text export settings dialog box, shown in Figure 5-3. To display this dialog box programmatically, you call the `MovieExportDoUserDialog` function.

Based on the options you specify in the text export settings dialog box, the text export component is assigned one of three text export option constants: `kMovieExportTextOnly`, `kMovieExportAbsoluteTime`, or `kMovieExportRelativeTime`.

Figure 5-3 Text export settings dialog box

If you choose “Show Text Only,” the text component is assigned the export option constant `kMovieExportTextOnly`. In this case, the text component exports only text samples, without text descriptors or time stamps. This option is useful when you want to export only the text from a movie and you do not intend to import the text back into a movie.

If you select “Show Text, Descriptors, and Time,” the text component is assigned one of two export option constants, depending on the format you specify for time stamps:

- If you specify time stamps to be relative to the start of the movie, the text component is assigned the export option constant `kMovieExportAbsoluteTime`. In this case, time stamps are calculated relative to the start of the movie. For example, in exported text with absolute time stamps, the time stamp [00:00:04.000] indicates that a text sample begins 4 seconds after the start of the movie.
- If you specify time stamps to be relative to the sample, the text component is assigned the export option constant `kMovieExportRelativeTime`. In this case, the time stamp for each sample is calculated relative to the end of the previous sample. For example, in exported text with relative time stamps, the time stamp [00:00:04.000] indicates that a text sample begins 4 seconds after the beginning of the previous sample. In other words, the previous sample lasts 4 seconds.

In both cases, text export component exports text, along with both text descriptors and time stamps. For more information about time stamps, see "Time Stamps" (page 51).

The text export component provides two functions you can use to access the component's text export option programmatically. To retrieve the current value of the text export option, you call `TextExportGetSettings`. To set the value of the text export option, you call `TextExportSetSettings`.

The Text Export Settings dialog box also allows you to specify the time scale the text component uses to specify the fractional part of a time stamp. The value should be between 1 and 10000, inclusive. The text export component provides two functions you can use to access the component's time scale programmatically. To retrieve the time scale, call `TextExportGetTimeFraction`. To set the time scale, call `TextExportSetTimeFraction`.

Time Stamps

When you export text and text descriptors from a text track, the text component also exports a time stamp for each sample. The time stamp indicates the starting and ending time of the sample, either relative to the start of the movie (`kMovieExportAbsoluteTime`) or to the end of the previous sample (`kMovieExportRelativeTime`). On import, the time stamps maintain the timing positions of the text samples relative to other media in the movie.

The format of a time stamp is

```
[HH:MM:SS.xxx]
```

where `HH` represents the number of hours, `MM` represents the number of minutes, `SS` represents the number of seconds, and `xxx` represents the mantissa (the fractional part of a second). The mantissa is expressed in the time scale of the text track. For example, if the time scale of the text track is 600, the time stamp [00:00:07.300] is interpreted as 7.5 seconds. If the time scale of the text track is 10, the time stamp [00:00:07.5] is also interpreted as 7.5 seconds. The maximum time scale for a text track is 10000.

When a text export component exports a text sample, it first exports the time stamp, followed by a return character. Then, it exports the sample's text and text descriptors. If a text sample does not contain any text, the text component exports the time stamp and return character, but no text.

Text Descriptors

A text descriptor is a formatting command that describes the text that follows it. Exporting text with text descriptors allows you to edit text from a text track, including its formatting, in an external program, such as a text editor or word processor. When you import the edited text, the formatting you specified with the text descriptors is preserved. This provides an easy way to localize movies for different languages, correct spelling, change styles, or modify text behavior.

A text descriptor has the format { *descriptor* }. For example, the text descriptor `{bold}` sets the text style in the current text sample and all subsequent text samples. Some text descriptors, such as `{bold}`, have no parameters. Other text descriptors have one or more parameters. For text descriptors with parameters, the descriptor is followed by a colon and its parameters, separated by commas. You can specify text descriptors using either uppercase or lowercase characters, with or without spaces separating the parameters:

{*descriptor: parameter1, ..., parameterN* }

For example, the text descriptor `{font:New York}` sets the text font in the current text sample and all subsequent text samples to the New York font. The New York font is applied to all text until a second `{font:}` text descriptor is issued.

A text stream that contains text descriptors and time stamps should always begin with the text descriptor `{QTtext}`, followed by any number of text descriptors in any order. If the text import component detects a typographical error inside a descriptor while importing a text file, it may generate partial results or an error message stating that the text file cannot be converted.

When text with text descriptors is imported into a track, the information represented by the descriptors is stored in a text display data structure (type `TextDisplayData`). Text descriptors whose possible values are `on` and `off` are used to set flags in the `displayFlags` field of the text display data structure. Each sample in the text track has a corresponding text display data structure that contains the text attributes of the sample. For more information, see "Text Display Data Structure" (page 53).

MIME Type List

The `MovieImportGetMIMETypeList` function returns a list of the MIME types supported by a movie import component. This list is contained in the QT atom container described in this section. For more information about QT atom containers, see the QuickTime Atom documentation.

At the top level of the atom container are three atoms for each supported MIME type. The atoms whose IDs are 1 describe the first supported MIME type, the atoms whose IDs are 2 describe the second supported MIME type, and so on.

An atom of type `kMimeInfoMimeTypeTag` contains a string that identifies the MIME type, such as `image/jpeg` or `image/x-jpeg`.

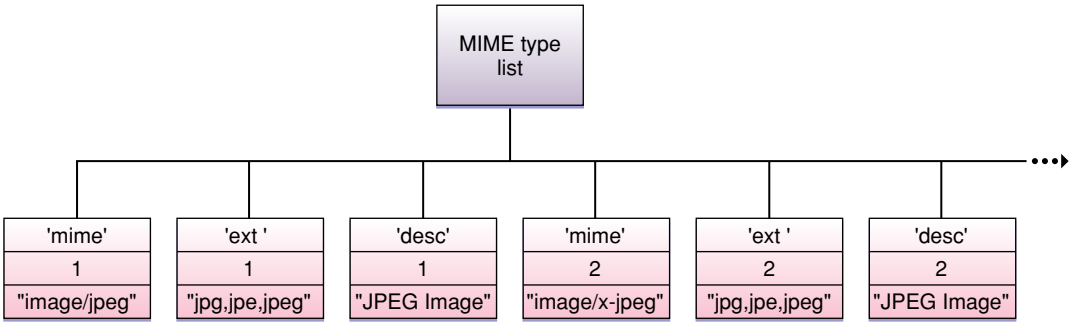
The atom of type `kMimeInfoFileExtensionTag` contains a string that specifies likely file extensions for files of this MIME type, such as `jpg`, `jpe`, and `jpeg`. If there is more than one extension, the extensions are separated by commas.

The atom of type `kMimeInfoDescriptionTag` contains a string describing the MIME type for end users, such as “JPEG Image.”

These atom types contain neither a Pascal nor a C string. The atom types are simply ASCII characters; an atom’s size is the number of characters. For best performance, include a public component resource of type `'mime'` and ID 1 with your exporter.

Figure 5-4 illustrates a MIME type list.

Figure 5-4 A MIME type list



Text Display Data Structure

The `TextDisplay` data structure contains formatting information for a text sample. When the text export component exports a text sample, it uses the information in this structure to generate the appropriate text descriptors for the sample. Likewise, when the text import component imports a text sample, it sets the appropriate fields in the text display data structure based on the sample's text descriptors.

```
struct TextDisplayData {
    long                    displayFlags;
    long                    textJustification;
    RGBColor            bgColor;
    Rect                    textBox;
    short                   beginHilite;
    short                   endHilite;
    RGBColor            hiliteColor;
    Boolean               doHiliteColor;
    SInt8                   filler;
    TimeValue           scrollDelayDur;
    Point                   dropShadowOffset;
    short                   dropShadowTransparency;
};
typedef struct TextDisplayData TextDisplayData;
```

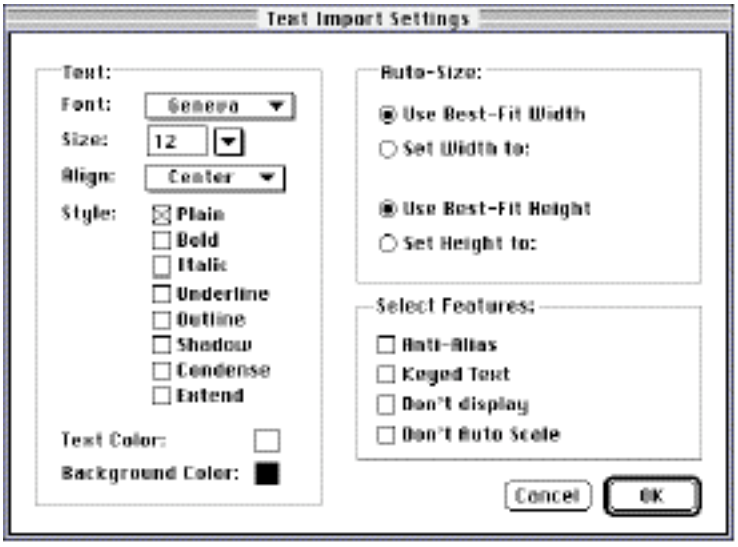
Term	Definition
`displayFlags`	Contains flags that represent the values of the following text descriptors: `doNotDisplay`, `doNotAutoScale`, `clipToTextBox`,`useMovieBackColor`, `shrinkTextBox`, `scrollIn`, `scrollOut`, `horizontalScroll`, `reverseScroll`, `continuousScroll`, `flowHorizontal`, `dropShadow`, `anti-alias`, `keyedText`, `inverseHilite`, `continuousKaraoke`, and `textColorHilite`.
`textJustification`	Specifies the alignment of the text in the text box. Possible values are `teFlushDefault`, `teCenter`, `teFlushRight`, and `teFlushLeft`.
`bgColor`	Specifies the background color of the rectangle specified by the `textBox` field. The background color is specified as an RGB color value.
`textBox`	Specifies the rectangle of the text box.

Term	Definition
`beginHilite`	Specifies the one-based index of the first character in the sample to highlight.
`endHilite`	Specifies the one-based index of the last character in the sample to highlight.
`doHiliteColor`	Specifies whether to use the color specified by the `hiliteColor` field for highlighting. If the value of this field is `true`, the highlight color is used for highlighting. If the value of this field is `false`, reverse video is used for highlighting.
`filler`	Reserved.
`scrollDelayDur`	Specifies a scroll delay. The scroll delay is specified as the number of units of delay in the text track's time scale. For example, if the time scale is 600, a scroll delay of 600 causes the sample text to be delayed one second. In order for this field to take effect, scrolling must be enabled.
`dropShadowOffset`	Specifies an offset for the drop shadow. For example, if the point specified is (3,4), the drop shadow is offset 3 pixels to the right and 4 pixels down. In order for this field to take effect, drop shadowing must be enabled.
`dropShadow-Transparency`	Specifies the intensity of the drop shadow as a value between 0 and 255. In order for this field to take effect, drop shadowing must be enabled.

Importing Text

When you import text, you can override the text descriptors in the text file by specifying options in the Text Import Settings dialog box, shown in Figure 5-5.

Figure 5-5 Text import settings dialog box



On import, the settings specified in the dialog box are applied to all imported samples. To display this dialog box programmatically, you can call the `MovieImportDoUserDialog` function.

Importing In Place

Some movie data import components can create a movie from a file without having to write to a separate disk file. Examples include MPEG, AIFF, DV, and AVI import components; data in files of these types can be played directly by the appropriate media handler components without any data conversion. In such cases it is inappropriate for the user to have to specify a destination file, because there is no need for such a file.

If your import component can operate in this manner, set the `canMovieImportInPlace` flag to 1 in your component flags when you register your component. The standard file dialog box uses this flag to determine how to import files. The `OpenMovieFile` and `NewMovieFromFile` functions use this flag to open some kinds of files as movies.

Audio CD Import Component

QuickTime includes an audio CD import component. This movie import component allows users to open audio CD tracks from the QuickTime standard file preview dialog box, then convert and save the audio as a movie.

When you open an audio track on an Apple CD-ROM drive (or equivalent), the Open button changes to a Convert button. When you click Convert, the audio CD import options dialog box appears. Use this dialog box to configure the sound settings. You can specify the sample rate, sample size, and channel settings. You can also select the portion of the track that you want to convert.

DV Video Import and Export Components

QuickTime includes movie data exchange components for DV video.

DV Movie Import Component

The DV movie import component converts a file containing DV video data into a QuickTime movie. The input file must be a Mac OS file of type `'dvc!'` or a Windows file with the .dv file extension. The output file contains a QuickTime movie with two tracks:

- A video track whose samples are of type `kDVNTSCCodecType` for NTSC video data or `kDVCPALCodecType` for PAL video data.
- A sound track whose samples are of type `kDVAudioFormat`.

The data is converted in place, as described in "Importing In Place" (page 55), and the import operation typically takes less than a second. Because both tracks in the QuickTime file refer to the same data, flattening the file creates a file that is twice the size of the original DV data.

You can perform the same operations on the resulting QuickTime movie (including playback, editing, and stepping) that you can for other QuickTime movies. Because video and audio are interleaved in the underlying data, applications for editing movies should make it possible to create a separate file that contains only the audio data for the movie.

This can be done by calling the `ConvertMovieToFile` function and specifying `kQTFileTypeAIFF` as the destination file format.

DV Movie Export Component

The DV movie export component converts a file containing a QuickTime movie to a file containing DV data for the movie.

The input file must contain a video track; the DV movie export component cannot convert a movie that contains only audio.

If the video track in the QuickTime movie is already in DV format, the DV movie export component does not recompress the video. This makes it possible to edit DV video in QuickTime and then export it without any loss of video quality due to recompression.

Exporting DV Data from an Application

An application can export DV data without creating a QuickTime movie file by using a callback procedure to supply media data, as described in "Exporting Data from Sources Other Than Movies" (page 56).

Exporting Data from Sources Other Than Movies

A movie data export component can be written to export data from sources other than QuickTime movies. To do this, the software that exports data must implement callback functions that provide services to the movie data export component. The callback functions and other functions that support this feature are described in this section.

The export component's `MovieExportFromProceduresToDataRef` routine performs data exporting. When executed, that routine makes callbacks to retrieve characteristics, called properties, and media data from each data source. Characteristics for a video data source might include the width, height, and image compression settings to be used; the media data would be the image description and image data corresponding to a particular movie time. A sound data source would have sound-related characteristics and sound sample data.

Follow these basic steps to export data:

- Find an exporter for the format that implements `MovieExportFromProceduresToDataRef`.
- Open that export component.
- For each data source, call the export component's `MovieExportAddDataSource` to provide the property and data callbacks that the exporter should call during export.
- Call `MovieExportFromProceduresToDataRef` to perform the export.

Determining What Kind of Tracks a Component Supports

The following code sample can be used to determine which track types a given movie export component supports.

Instantiating the Data Export Component

The first step in using a movie data export component to create an AIFF file is instantiating an AIFF data export component. An example of this is shown in Listing 5-1.

Listing 5-1 Instantiating a data export component

```
QTAtomContainer container = nil;
ComponentDescription cd;
long count, i;
cd.componentType = MovieExportType;
cd.componentSubType = 0;
cd.componentManufacturer = 0;
cd.componentFlags = 0;
cd.componentFlagsMask = 0;
GetComponentPublicResourceList(kQTMovieExportTrackInfoResourceType, 1, 0, &cd,
 nil, nil, &container);
count = QTCountChildrenOfType(container, kParentAtomIsContainer,
OSTypeConst('comp'));
for (i=1; i<=count; i++) {
    QTAtom compAtom, trkResourceAtom;
    Component c;
    compAtom = QTFindChildByIndex(container, kParentAtomIsContainer,
        OSTypeConst('comp'), i, (long*)&c);
    trkResourceAtom = QTFindChildByID(container, compAtom,
        kQTMovieExportTrackInfoResourceType, 1, nil);
    if (trkResourceAtom) {
        QTMovieExportSourceRecord **trkResource = (QTMovieExportSourceRecord**)
            NewHandle(0);
        long j;
        Boolean wantsSound = false, wantsVideo = false, wantsText = false,
            wantsMIDI = false;
       QTCopyAtomDataToHandle(container, trkResourceAtom, (Handle)trkResource);
        for (j=0; j<(**trkResource).count; j++) {
            OSType mType = (**trkResource).sourceArray[j].mediaType;
            long flags = (**trkResource).sourceArray[j].flags;
            wantsSound = wantsSound ||
                ((mType == SoundMediaType) && (flags &
                    kQTMovieExportSourceInfoIsMediaType));
            wantsVideo = wantsVideo ||
                (((mType == VideoMediaType) && (flags &
                    kQTMovieExportSourceInfoIsMediaType)) ||
                ((mType == VisualMediaCharacteristic) && (flags &
                    kQTMovieExportSourceInfoIsMediaCharacteristic)));
            wantsText = wantsText ||
                ((mType == TextMediaType) && (flags &
                    kQTMovieExportSourceInfoIsMediaType));
            wantsMIDI = wantsMIDI ||
                ((mType == MusicMediaType) && (flags &
```

```
                    kQTMovieExportSourceInfoIsMediaType));
        }
        if (wantsSound || wantsVideo) {
            ComponentDescription cd;
            char cc[5];
            GetComponentInfo(c, &cd, nil, nil, nil);
            cc[0] = 4;
            *(long *)&cc[1] = cd.componentSubType;
            DebugStr((StringPtr)cc);
        }
        DisposeHandle((Handle)trkResource);
    }
}
QTDisposeAtomContainer(container);
```

Using a Movie Data Export Component to Export Audio

Listing 1-2 illustrates how to use a movie data export component to export audio data to an AIFF file.

Listing 5-2 Exporting audio data to an AIFF file

```
ComponentDescription cd;
MovieExportComponent ci;
cd.componentType = MovieExportType;
cd.componentSubType = 'AIFF';
cd.componentManufacturer = SoundMediaType;
cd.componentFlags = canMovieExportFromProcedures;
cd.componentFlagsMask = canMovieExportFromProcedures;
ci = OpenComponent(FindNextComponent(nil, &cd));
```

Note that the `componentManufacturer` field holds the

`SoundMediaType` if the movie contains sampled sound or `'musi'` if it contains MIDI music. If you pass a zero in this field, QuickTime will use the first exporter that can create the desired type of output. This will not always produce the desired result, as a component that creates AIFF output from MIDI requires different input that a component that creates AIFF output from sampled sound.

Note that you use the `canMovieExportFromProcedures` flag to limit the search to exporters that support the `MovieExportFromProceduresToDataRef` component call This is important since not all exporters implement this routine.

Configuring the Data Export Component

Once an AIFF movie data export component has been instantiated, it must be configured to open a single output audio stream. Listing 5-3 is an example of creating an output audio stream by calling `MovieExportAddDataSource`. In this example, `MovieExportAddDataSource` also provides the callback functions for supplying media data.

Listing 5-3 Configuring the audio export component

```
#define kMySampleRate 22050
#define kSoundBufferSize 1024
```

```
typedef struct
{
    Ptr soundData;
    SoundDescriptionHandle soundDescription;
    long trackID;
}
MyReferenceRecord;
MyReferenceRecord myRef;
SoundDescriptionPtr sdp;
myRef.soundData = NewPtr(kSoundBufferSize);
myRef.soundDescription = NewHandleClear(sizeof(SoundDescription));
sdp = *myRef.soundDescription;
sdp->descSize = sizeof(SoundDescription);
sdp->dataFormat = k8BitOffsetBinaryFormat;
sdp->numChannels = 1;
sdp->sampleSize = 8;
sdp->sampleRate = kMySampleRate << 16;
MovieExportAddDataSource(ci, SoundMediaType, kMySampleRate,
        &myRef.trackID, getSoundTrackPropertyProc,
        getSoundTrackDataProc, &myRef);
```

On the Macintosh, the getSoundTrackPropertyProc and getSoundTrackDataProc routines should be universal procedure pointers (UPPs).

Exporting the Data

The export operation takes place when all of the required output tracks have been created. Typical code is shown in Listing 5-4.

Listing 5-4 Exporting from procedures to a data reference

```
StandardFileReply reply;
Handle dataRef;
// get output file from user
QTNewAlias(&reply.sfFile, (AliasHandle *)&dataRef, true);
// make up a data reference
MovieExportFromProceduresToDataRef(ci, dataRef, rAliasType);
```

`MovieExportFromProceduresToDataRef` calls the two functions specified in `MovieExportAddDataSource` to obtain data to generate the output file. The first function returns information about the output track's properties, including the sample rate and supported media. If no value is returned for a particular property, the exporter specifies a default value based on the source data format. In the example in Listing 5-5, the output sample rate is set at 32000 Hz, with all other properties left unspecified.

Listing 5-5 Obtaining output track information

```
pascal OSErr getSoundTrackDataProc(void *refcon, long trackID,
    OSType propertyType, void *propertyValue)
{
    OSErr err = noErr;
    switch (propertyType)
    {
        case scSoundSampleRateType:
            *(Fixed *)propertyValue = 32000L << 16;
            break;
```

```
        default:
            err = paramErr;
            break;
    }
    return err;
}
```

The second function provides data to be exported.

Listing 5-6 shows a block of sound data (silence) returned for each export request. The export operation ends when this function returns `eofErr`.

Listing 5-6 Providing output track information to the export component

```
pascal OSErr getSoundTrack(void *refCon,
        MovieExportGetDataParams *params)
{
    MyReferenceRecord *myRef = (MyReferenceRecord *)refCon;
    if (params->requestedTime > kMySampleRate * 10)
        return eofErr;                  // end of data after 10 seconds
    params->dataPtr = myRef->soundData;
    params->dataSize = kSoundBufferSize;
    params->actualTime = params->requestedTime;
    params->sampleCount = kSoundBufferSize;
    params->durationPerSample = 1;
    params->descType = SoundMediaType;
    params->descSeed = 1;
    params->desc = (SampleDescriptionHandle)myRef->soundDescription;
    return noErr;
}
```

Using a Movie Data Export Component to Export Video

Using a movie data export component to create a QuickTime movie file is similar in many respects to creating an AIFF file, as shown in the previous example. Media data is handled differently in each case, however.

Instantiating the Video Export Component

Listing 5-7 illustrates the first step, instantiating the movie data export component for video data.

Listing 5-7 Instantiating a movie data export component

```
ComponentDescription cd;
MovieExportComponent ci;
cd.componentType = MovieExportType;
cd.componentSubType = 'MooV';
cd.componentManufacturer = 'appl';
cd.componentFlags = canMovieExportFromProcedures;
cd.componentFlagsMask = canMovieExportFromProcedures;
ci = OpenComponent(FindNextComponent(nil, &cd));
```

Configuring the Video Export Component

Listing 5-8 illustrates the next step: configuring the data export component to create a single output video track. The call to `MovieExportAddDataSource` provides the callback functions for supplying media data.

Listing 5-8 Configuring the movie data export component

```
#define kMySampleRate 2997      // 29.97 fps
#define kMyFrameDuration 100    // one frame at 29.97 fps
typedef struct
{
    GWorldPtr gw;               // temporary GWorld we use during export
    ImageDescriptionHandle imageDescription;
    long trackID;
}
MyReferenceRecord;
MyReferenceRecord myRef;
myRef.gw = nil;
myRef.imageDescription = nil;
MovieExportAddDataSource(ci, VideoMediaType, kMySampleRate,
        &myRef.trackID, getVideoPropertyProc,
        getVideoDataProc, &myRef);
```

Using the Component

The foregoing process should be repeated for as many video and sound sources as will be exported. However, it's important to realize that not all exporters supporting export from procedures can export an arbitrary number of sources. For the case where an exporter supports fewer sources than the application needs to export, the application must precomposite the video sources or mix the sound sources before providing them to the exporter. The export code was shown previously in Listing 1-4.

The `getVideoPropertyProc` function returns information about the output track's properties (for example, the compression format). If the function doesn't return a value for a particular property, the exporter will choose a default value based (usually) on the source data format. In Listing 5-9, dimensions are set to 160x120 and the compression format is set to JPEG. All other properties are unspecified.

Listing 5-9 Setting dimensions and compression format

```
pascal OSErr getVideoPropertyProc(void *refcon, long trackID,
    OSType propertyType, void *propertyValue)
{
    OSErr err = noErr;
    switch (propertyType) {
        case meWidth:
            *(Fixed *)propertyValue = 160L << 16;
            break;
        case meHeight:
            *(Fixed *)propertyValue = 120L << 16;
            break;
        case scSpatialSettingsType:
            {
            SCSpatialSettings *ss = propertyValue;
            ss->codecType = kJPEGCodecType;
            ss->codec = 0;
```

```
            ss->depth = 0;
            ss->spatialQuality = codecNormalQuality;
            }
            break;
        default:
            err = paramErr;
            break;
    }
    return err;
}
```

The `videoGetDataProc function` provides video frames to the export operation. In the example in Listing 5-10, the same blank frame is returned for each request. The export operation ends when this function returns `eofErr`. Any data allocated by `videoGetDataProc must` be disposed of after the export operation is complete.

Listing 5-10 Providing video frames for export

```
pascal OSErr getVideoDataProc(void *refCon,
        MovieExportGetDataParams *params)
{
    OSErr err = noErr;
    MyReferenceRecord *myRef = (MyReferenceRecord *)refCon;
    TimeRecord tr;
    if (params->requestedTime > kMySampleRate * 10)
        return eofErr;// end of data after 10 seconds
    if (!myRef->gw) {
        Rect r;
        CGrafPtr savePort;
        GDHandle saveGD;
        SetRect(&r, 0, 0, 320, 240);
        NewGWorld(&myRef->gw, 32, &r, nil, nil, 0);
        LockPixels(myRef ->gw->portPixMap);
        MakeImageDescriptionForPixMap(myRef->gw->portPixMap,
                    &myRef->imageDescription);
        GetGWorld(&savePort, &saveGD);
        SetGWorld(myRef->gw, nil);
        EraseRect(&r);
        SetGWorld(savePort, saveGD);
    }
    params->dataPtr = GetPixBaseAddr(myRef->gw->portPixMap);
    params->dataSize = (**myRef->imageDescription).dataSize;
    params->actualTime = params->requestedTime;
    params->descType = VideoMediaType;
    params->descSeed = 1;
    params->desc = (SampleDescriptionHandle)
                    myRef->imageDescription;
    params->durationPerSample = kMyFrameDuration;
    params->sampleFlags = 0;
}
```

Determining the Data Sources Supported by an Export Component

The number and kind of data sources supported varies from one exporter to another. In the foregoing examples, AIFF only supports sound data sources while the QuickTime movie exporter accepts both sound and video data sources. Furthermore, an exporter may require at least one data source of some type. In others, it may accept 0 to some maximum number of data sources of a particular type.

Because the kinds and number of data sources supported varies from exporter to exporter, and may change with new versions of a particular exporter, there needs to be a way for a client of an exporter to determine this information. In QuickTime, a procedure-supporting exporter has a public component resource that provides this information.

Creating a Movie Data Exchange Component

This chapter discusses the details of creating a movie data exchange component. The chapter also 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.

Before reading this chapter, you should be familiar with how to create components. After reading this chapter, you should understand the special requirements of these components. Note that a single component may support only import or export functions, not both.

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

Apple has defined a functional interface for movie data exchange components. You can use the following constants to refer to the request codes for each of the functions that your component must support:

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

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 */
    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 */
};
```

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" (page 82) for more information about these functions). However, your component should work properly even if none of these configuration functions is called.

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" (page 82) for more information about these functions). However, your component should work properly even if none of these configuration functions is called.

Summary of 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 */
    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 */
};
```

Result Codes

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

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 supplies a sample program that implements a movie data exchange component that imports a Scrapbook file containing QuickDraw PICT images.

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 or `MovieImportFile` function.

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 6-1 supplies a sample program that implements a movie data exchange component that imports a Scrapbook file containing QuickDraw PICT images. The sample program also provides the dispatchers for the movie import component together with the required functions.

Listing 6-1 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;
    }

    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)
{
    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.

The `MovieImportFile` function 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 6-2 supplies an example in which a Scrapbook file is imported.

Listing 6-2 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;

    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);

/* 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;

    /* 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, 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 6-3 provides the component dispatchers for the movie export component together with the required functions.

Listing 6-3 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;
}

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 6-4 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 6-4 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;
```

```
        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;
}
```

Save-and-Restore Component Routines

If you are writing movie data exchange components, and would like your components' settings to be saved and restored, you need to implement two additional component routines in order to allow your components to have their settings saved and restored. A component's settings are stored in a QuickTime QT atom container. The data stored in the QT atom container is private to the particular component but should be stored so that it is possible to read the data on all platforms supported by QuickTime, thus allowing the same settings to be used anywhere.

For each type of movie data exchange component, there is one routine to return a QT atom container holding the settings and another routine to configure the component from previously-saved settings. For more information about QT atom containers, see *QuickTime Movie Basics*.

Import component developers need to implement the `MovieImportGetSettingsAsAtomContainer` and `MovieImportSetSettingsFromAtomContainer` routines. For `MovieImportGetSettingsAsAtomContainer`, the component should allocate a new QT atom container, stuff current settings into it, and return it to the caller. For `MovieImportSetSettingsFromAtomContainer`, the component should accept a QT atom container, extract the settings in which it is interested, and change its internal state.

Export component developers need to implement the `MovieExportGetSettingsAsAtomContainer` and `MovieExportSetSettingsFromAtomContainer` routines. Like import components, the component's `MovieExportGetSettingsAsAtomContainer` routine allocates and returns a QT atom container holding the component's settings. For `MovieExportSetSettingsFromAtomContainer`, the component accepts a QT atom container, extracts the settings, and updates its internal state.

Settings Container Format and Guidelines

The particular atoms stored within the component's settings QT atom container are private to that component type. However, there are some guidelines that need to be followed. These include:

In all `SetSettingsFromAtomContainer` routines, the QT atom container belongs to the caller. The component should not dispose of the passed QT atom container.

The settings QT atom container should contain one or more top-level atoms. These top-level atoms can contain either leaf data or other atoms. Each atom has both a type (`QTAtomType`) and an ID. Choosing an atom type that is mnemonic is helpful in indicating how it is used. For example, QuickTime stores video compression settings in atoms of type `'vide'`. Sound compression settings are stored in `'soun'` atoms. The text components use `'text'` for their atom types.

Several of QuickTime's export components use the standard compression component to allow the user to configure compression settings for exported files. When one of these components is asked to return its settings atom container, the export component first requests that the standard compression component return its settings using the `SCGetSettingsAsAtomContainer` function described above. To the QT atom container it receives, the export component adds any of its own settings. When the export component's `SetSettingsFromAtomContainer` is called, the exporter calls

`SCSetSettingsFromAtomContainer` with the passed atom container. The standard compression component extracts only those settings it expects, ignoring all other, and configures itself. The exporter then looks for its own settings in the same atom container and configures itself.

This is possible because the standard compression and data exchange components both use QT atom containers to hold their settings. Because many third-party developers do the same, there must be a mechanism so that QuickTime's own top-level atom types and those of third parties don't collide. To achieve this, Apple Computer reserves all top-level atom types consisting exclusively of lowercase letters with or without numerals. For example, `'vide'` is reserved by Apple, but `'Vide'` is not. There is no restriction on the atom types for atoms stored within these top-level atoms.

Apple recommends that you store all of your component settings under a single top-level atom. However, there is no requirement to do so.

The data within an atom should be stored in a canonical form on all platforms. It should be always in big-endian format or always in little-endian format. Different types of atoms can be stored in different endian orders but for a single type of atom, it should always use the same order. This allows the settings to be created in the Mac OS and read in Windows or vice-versa.

In either `MovieImportSetSettingsFromAtomContainer` or `MovieExportSetSettingsFromAtomContainer`, you should not necessarily expect all atoms to be included in the atom container you receive. This allows another developer, for example, to create a settings atom container, add atoms and data for only those parts of the settings that should be changed, and then pass this incomplete atom container to the component. The component then only changes those particular settings, leaving other settings alone. QuickTime's own components use this approach.

If `nil` is passed for the settings to the component routines, return `paramErr`.

If your component does not have configurable settings, you do not need to implement the settings-related routines.

Registering Movie Data Export Components

QuickTime allows more than one export (`'spit'`) component to be registered for the same type of file and the same export source (the movie or the particular track type). This is accomplished in a way that preserves compatibility with third-party components that may have already been written using the former rules.

The Registration Mechanism

QuickTime provides a movie export component routine that returns the same information that would have been previously stored in the `componentManufacturer` field of the registered `'spit'` components. An export-specific component flag indicates that the export component implements the new protocol. By implementing the routine, the export component's `componentManufacturer` field can be used to differentiate components.

The routine is `MovieExportGetSourceMediaType`. This routine returns an `OSType` value through its `mediaType` parameter, which is interpreted in exactly the same way that the `componentManufacturer` was previously interpreted. If the export component requires a particular type of track to exist in a movie, it returns that media handler type (e.g., `VideoMediaType`, `SoundMediaType`, etc.) through the `mediaType` argument. If the export component works for an entire movie, it returns 0 through this parameter.

```
EXTERN_API( ComponentResult )
MovieExportGetSourceMediaType (MovieExportComponent  ci, OSType * mediaType);
```

The following component flag indicates that this routine is implemented:

```
movieExportMustGetSourceMediaType = 1L << 19,
```

If you implement the `MovieExportGetSourceMediaType` routine, you must register the component with this flag. Otherwise, the Movie Toolbox will not know to call the routine and will assume the older semantics for the `componentManufacturer` field.

Using this mechanism does not replace the need for implementing `Validate` in your export components. The mechanism is only used to find candidate components.

Export Registration Mechanism

Both the Movie and the DVC export components use the export registration mechanism. The components are registered as shown below.

```
componentType  'spit'
componentSubType  'MooV'
componentManufacturer  'appl'
componentFlags  canMovieExportFiles  + canMovieExportFromProcedures  +
                hasMovieExportUserInterface   + canMovieExportValidateMovie
                + movieExportMustGetSourceMediaType

componentType 'spit'
componentSubType   'dvc!'
componentManufacturer 'appl'
componentFlags  canMovieExportFiles  + canMovieExportFromProcedures
                + hasMovieExportUserInterface  + canMovieExportValidateMovie
                + movieExportMustGetSourceMediaType
```

Because the DVC component uses the QuickTime Movie export component, it searches for the 'MooV' exporter, using the following `ComponentDescription` values:

```
cd.componentType = 'spit';
cd.componentSubType = MovieFileType;
cd.componentManufacturer = 'appl';
cd.componentFlags = canMovieExportFromProcedures
                    + movieExportMustGetSourceMediaType;
cd.componentFlagsMask = cd.componentFlags;
```

If you are working with export components (either writing them, or trying to enumerate or otherwise match up components with source media types) you need to understand this registration mechanism.

Implementing Movie Data Export Components

You can implement a movie data export component by calling the `MovieExportFromProceduresToDataRef` function.

Because many existing applications expect to be able to perform an export operation from a movie or track, export components should support `MovieExportToFile`, `MovieExportFromProceduresToDataRef` and `MovieExportToDataRef`.

Listing 6-5 shows how to implement `MovieExportToFile` so that it simply calls `MovieExportToDataRef`.

Listing 6-5 Calling MovieExportToDataRef from MovieExportToFile

```
pascal ComponentResult MovieExportToFile(Globals store, const FSSpec *theFile,
 Movie m, Track onlyThisTrack, TimeValue startTime, TimeValue duration) {
ComponentResult err;
     AliasHandle alias;
     err = QTNewAlias (theFile, &alias, true);
     err = MovieExportToDataRef(store->self, (Handle)alias, rAliasType, m,
onlyThisTrack, startTime,
```

```
            duration);
        DisposeHandle((Handle)alias); }
```

Listing 6-6 shows how to use the utility routines provided by the QuickTime movie data export component to implement `MovieExportToDataRef` by calling `MovieExportFromProceduresToDataRef`. Your implementation may differ, depending on the types of data to be exported. For example, the number and type of data sources created may change. This example creates a single sound data source, and is appropriate for any movie data export component that exports audio only.

Listing 6-6 Calling MovieExportFromProceduresToDataRef from MovieExportToDataRef

```
pascal ComponentResult MovieExportToDataRef(Globals store, Handle dataRef,
  OSType dataRefType, Movie m, Track onlyThisTrack, TimeValue startTime,
  TimeValue duration) {
     ComponentResult err;
     ComponentDescription cd;
     ComponentInstance ci;
     TimeScale scale;
     MovieExportGetPropertyUPP getPropertyProc = nil;
     MovieExportGetDataUPP getDataProc = nil;
     void *refCon;
     long trackID;
     cd.componentType = MovieExportType;
     cd.componentSubType = 'MooV';
     cd.componentManufacturer = 0;
     cd.componentFlags = canMovieExportFromProcedures;
     cd.componentFlagsMask = canMovieExportFromProcedures;
     err = OpenAComponent(FindNextComponent(nil, &cd), &ci);
     err = MovieExportNewGetDataAndPropertiesProcs(ci, SoundMediaType,
           &scale, m, onlyThisTrack,  startTime, duration, &getPropertyProc,
           &getDataProc, &refCon);
     err = MovieExportAddDataSource(store->self, SoundMediaType, scale,
           &trackID, getPropertyProc, getDataProc, refCon);
    err = MovieExportFromProceduresToDataRef(store->self, dataRef, dataRefType);
}
```

The code in Listing 6-6 retrieves default property and data procedures, instead of providing them, by using the QuickTime Movie export component. It also must dispose of these procedures. They take care of interpreting the tracks and returning media properties and data.

Using Movie Data Exchange Components

This chapter describes how to use movie exchange components from within your application. The chapter focuses on three particular features: the ability to specify part of a file to import, getting a list of supported MIME types, and determining whether movie data export is possible.

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.

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" (page 82) and "Configuring Movie Data Export Components" (page 82) 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`.

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 `MovieImportSetProgressProcfunction`.

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

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.

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 component. 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. 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.

Specifying a Part of a File to Import

When using certain movie import components, applications can import data from a part of a file rather than the entire file by calling `MovieImportSetOffsetAndLimit` or `MovieImportSetOffsetAndLimit64`. The latter function accomodates 64-bit offsets instead of just 32-bit offsets. These functions let an application specify a byte offset into the file at which the import operation begins and another offset, known as the *limit,* that indicates the last data in the file that can be imported. This function is especially useful when one file format is embedded in another; it allows an application to skip header data for the enclosing file and begin importing data at the start of the desired format.

Not all movie import components support the `MovieImportSetOffsetAndLimit` or `MovieImportSetOffsetAndLimit64` function. Those that do include the movie import components provided with QuickTime for the `kQTFileTypeAIFF`, `kQTFileTypeWave`, and `kQTFileTypeMuLaw` file types. Those that do not return the result code `badComponentSelector` in response to a `MovieImportSetOffsetAndLimit` or `MovieImportSetOffsetAndLimit64` call. If your export component implements `MovieImportSetOffsetAndLimit64`, it should implement `MovieImportSetOffsetAndLimit` too, since older clients may not use the new 64-bit offset version of the call.

Getting a List of Supported MIME Types

Applications can get a list of MIME types supported by a movie import component by calling the `MovieImportGetMIMETypeList` function.

In QuickTime, import components should additionally include a public component resource holding the same data that `MovieImportGetMIMETypeList` would return. This public resource's public type and ID should be `'mime'` and 1, respectively; these values are held in the import component's public resource list. The following shows an example of such a list:

```
resource 'thnr' (kMyImportComponentResID)
{
    'mime', 1, 0,
    'mime', kMyImportMIMETypeListResID, 0
}
```

By including this public resource, QuickTime and applications don't need to open the import component and call `MovieImportGetMIMETypeList` to determine the MIME types the importer supports. In the absence of this resource, QuickTime and applications will use `MovieImportGetMIMETypeList`.

Determining Whether Movie Data Export Is Possible

Although a movie export component can export one or more media types, it may not be able to export all the kinds of data stored in those media. Applications can find out whether a movie export component can export all the data for a particular movie or track by calling the `MovieExportValidate` function.

Not all export components implement the `MovieExportValidate` call. In the following code snippet, you make the `Validate` call, and even if the component routine is not implemented it is still true:

```
Boolean canExport = true;
```

```
MovieExportValidate(ci, &canExport);
if(canExport) {
. . .
}
```

Document Revision History

This table describes the changes to *QuickTime Import and Export Guide*.

Date	Notes
2006-01-10	New document that describes QuickTime's technology for importing and exporting graphics and other data into and out of movies.
	Replaces "Movie Import/Export (Data Exchange Components)," "Graphics Importers," and "Graphics Exporters."
2002-09-17	New document that describes the components that transport data between non-movie formats and QuickTime movies.

Document Revision History