QT 02 - Inside Macintosh: QuickTime Components Addendum

QuickTime

Written by: Developer Technical Support and QuickTime Engineering December 1994

This Tech Note is an addendum to the Inside Macintosh: QuickTime Components publication. It will contain technical details of QuickTime missing in the documentation, updated information, known problems, workarounds, bug fixes and similar information. The subtitles are based on the QuickTime Components Publication with the addition of new ones related to additional information not present in the documentation.

We assume that developers use QuickTime 1.6.1 or QuickTime 2.0, any older versions are no longer supported by DTS.

Table of Contents

CHAPTER 2 - MOVIE CONTROLLER COMPONENTS

Standard Controller with MCCut or MCClear

Q: When I select all frames in QuickTime and then do an MCCut or MCClear, the standard controller gets larger and redraws itself at the top of the movie. Is this a situation I should be prepared to handle or a bug? Does the controller behave strangely when the selectionTime of a movie is -1 or when the duration of the movie is 0?

___

A: The behavior you're observing is to be expected if the controller is attached to the movie. In this case, the controller goes to wherever the bottom left corner of the movie box takes it. If the movie loses all its "visible" parts, the movie controller will jump to the top of the window. The only way to get around this is to detach the controller when the movie box is empty; this is also something to keep in mind for the cases when the movie contains only sound, since pure sound movies have no dimensions. You can find sample code showing how to do this on the *Developer CD Series* disc, in the SimpleInMovies example that accompanies the QuickTime article in *develop* Issue 7.

Techniques for Controller to Go to Beginning or End

Q: The controller constants have been left out of the latest Movies.h header for Goto beginning and ending commands. Only mcGotoTime is left. Is it OK to use these constants or is there a newly defined and accepted way to do this now?

___

Q: Scanning the interfaces, we can't locate a time when the controller had actions to go to the beginning or end. In any case there are two ways of accomplishing the same objective: The first is to use the MCActionGotoTime and pass the zero to go to the start of the movie and the result from GetMovieDuration to go to the end of the movie. The code to do this gets a little bit cumbersome since you have to pass a TimeRecord. This may be why our favorite is the second method.

Second, to go to the start, call GoToBeginningOfMovie followed by MCMovieChanged. And to go to the end, call GoToEndOfMovie also followed by MCMovieChanged. The following routine shows how to accomplish what you want:

```
void DoGotoEnds(short direction)
{
WindowPtr window;
DocRecHandle wHndl;

  if (window = FrontWindow()) { /* don't bother if no movies to play */
    if (IsAppWindow(window) && (wHndl = (DocRecHandle)GetWRefCon(window)) ) {
      switch (direction) {
        case gotoBeginning:
          GoToBeginningOfMovie( (*wHndl)->wMovie);
          break;
        case gotoEnd:
          GoToEndOfMovie( (*wHndl)->wMovie );
          break;
        default:
          return; /* do nothing if came here on error */
          break;
      }
    MCMovieChanged ((*wHndl)->wPlayer, (*wHndl)->wMovie);
    }
  }
}
```

Doing this you let the Movie Toolbox take care of all the details and the results are the same. MCMovieChanged is needed in order to make the controller aware of the different state.

As a bonus, the following code shows (in a different scenario) how you can fill in the time record necessary for the MCActionGoto call in case that’s the route you want to take:

```
/* selects the whole movie for editing */
void DoSelectAll(void)
{
WindowPtr window;
DocRecHandle wHndl;
ComponentResult err;
TimeRecord tRec;

  if (window = FrontWindow()) { /* don't bother if no movies to play */
    if (IsAppWindow(window) && (wHndl = (DocRecHandle)GetWRefCon(window)) ) {
      if ( MCIsEditingEnabled((*wHndl)->wPlayer)) {
        tRec.value.lo = tRec.value.hi = 0;
        tRec.base = 0;
        tRec.scale = GetMovieTimeScale((*wHndl)->wMovie);
        if ( err = MCDoAction((*wHndl)->wPlayer, mcActionSetSelectionBegin, &tRec)) {
          DebugStr("\pError trying mcActionSetSelectionBegin");
        }

        tRec.value.hi = 0;
        tRec.value.lo = GetMovieDuration((*wHndl)->wMovie);
        tRec.base = 0;
        tRec.scale = GetMovieTimeScale((*wHndl)->wMovie);
        if ( err = MCDoAction((*wHndl)->wPlayer, mcActionSetSelectionDuration, &tRec)) {
          DebugStr("\pError trying mcActionSetSelectionDuration");
        }
      }
    }
  }
}
```

Palettes, MCDoAction, and mcFlagsUseWindowPalette

If you use standard movie controllers, you need to make sure that the window palette is used. The following code example shows how this is done:

```
   MCDoAction(mcPlay, mcActionGetFlags, &controllerFlags);
   MCDoAction(mcPlay, mcActionSetFlags, (void *)(controllerFlags |
                 mcFlagsUseWindowPalette));
```

If you don't use movie controllers, you need to use the GetMovieColorTable function and supply the palette to the Palette Manager.

Note that MoviePlayer handles window palettes; you could always compare how the movie is playing using MoviePlayer and compare its palette behavior with the behavior of your application playing the same movie.

Drag Manager Disabling with Movie Controllers

In QuickTime 2.0 it is possible to disable dragging from the movie (Drag Manager), as in:

```
       MCDoAction(aController, mcActionSetDragEnabled, (void *)false);
```

This disables dragging from the movie , but it does not prevent the user from dragging another movie and dropping it on the movie for which dragging was disabled.

The workaround is to create a small offscreen GWorld, then call MCSetControllerPort to set the movie into that port. Use MCDoAction to turn off dragging, then use MCSetControllerPort to reset the movie to its original display. Remember to test for QuickTime 2.0 before doing this.

Here's an example that, given a movie controller, will disable the Drag Manager support:

```
OSErr  DoIgnoreDrags( MovieController  aController )
{
   GWorldPtr  tempGWorld;
   Rect       tempRect = {0, 0, 20, 20};
   OSErr      myErr = noErr;
   CGrafPtr   port;

   // The following code gets around a QuickTime 2.0 problem
   // with the drag manager.

   // Set up so we ignore drags.  First create a 1-bit small 20x20
   // offscreen:
   myErr = NewGWorld( &tempGWorld, 1, &tempRect, nil, nil, 0L );

   if(myErr != noErr)
   {
       // get the current port
       port = MCGetControllerPort( aController );

       // set the movie controller port to the new offscreen
       MCSetControllerPort( aController, (CGrafPtr)tempGWorld );

       // we don't want drags
       MCDoAction( aController, mcActionSetDragEnabled, (void *)false);

       // restore the movie controller port
        MCSetControllerPort( aController, port );

       // dispose the offscreen
       DisposeGWorld( tempGWorld );
   }
   return myErr ;
}
```

Problems with Updating the MovieController when Controlling the Start of a Movie

Q: When I control the start of a QuickTime movie from within my application, the movie controller doesn’t get updated properly. I’m calling StartMovie to begin the movie as soon as it becomes visible, and I’m updating the movie controller like this:

```
MCDoAction(myMovieController, mcActionPlay, &theRate);
```

However, this doesn’t seem to work. What am I doing wrong?

A: The MCDoAction call with mcActionPlay doesn’t take a pointer to the data in the last parameter; it takes the data itself. But since the prototype specifies type (void *), to make the compiler accept the code it must be cast to a pointer.

The recommended method to start a movie when you’re using the standard movie controller component is as follows:

```
Boolean PlayMovie(Movie theMovie, MovieController mc)
{
// Play normal speed forward, taking into account the possibility
// of a movie with a nonstandard PreferredRate.
       Fixed  aRate;
       OSErr  anErr;

       aRate= GetMoviePreferredRate(theMovie);
       anErr = DoPrerollMovie(theMovie);  // Important: Preroll the movie here.

       if(anErr == noErr)
       {
              MCDoAction(mc, mcActionPlay, (void *)aRate);  // note last value
              return true;
       }
       else
              return false;
}
```

```
OSErr DoPrerollMovie(Movie theMovie)
{
       TimeValue            aTimeValue;
       TimeValue            aMovieDur;
       Fixed                aPreferredRate;
       OSErr                anErr = noErr;

       aTimeValue      = GetMovieTime(theMovie, nil);
       aMovieDur       = GetMovieDuration(theMovie);
       aPreferredRate  = GetMoviePreferredRate(theMovie);

       anErr = PrerollMovie(theMovie, aTimeValue, aPreferredRate);

       return anErr;
}
```

If you do need to use StartMovie, the correct way to cause the movie controller to update is to call MCMovieChanged.

Problems with Movie Controller and Badge Outside a Frame

Q: We're trying to display a QuickTime movie in a frame that can be panned, cropped, and overlaid by other objects. The movie controller doesn't seem to understand that the badge may lie entirely outside the frame. Is there some way to tell the movie controller where to place the badge?

A: Unfortunately, QuickTime isn't flexible about this. The code that positions the badge calculates it from the bounding box of the movie region, and insets it six pixels from the left and bottom. There's no good way to work around this other than not to use the standard badge, but instead use your own badge and perform your own badge tracking.

CHAPTER 3 - STANDARD IMAGE -COMPRESSION DIALOG COMPONENTS

Standard Image-Compression Dialog Component Functions

Setting Default Values Without Using the Standard Compression Dialog

Q: I've noticed some interesting behavior using the standard compression dialog and was wondering if someone could explain it to me. I'm trying to provide session-wide defaults for compressing sequences of images. If I don't prime the dialog by doing an SCRequestSequenceSettings, then when I do an SCCompressSequenceBegin the dialog is displayed. Is there any way to prevent this, and to use some set of defaults (without using an image to derive the defaults)?

___

A: The compression dialog components allow you to get settings with the SCGetInfo call and to set them with SCSetInfo. The first time, you should display the dialog with SCRequestSequenceSettings, and then use SCGetInfo to retrieve the settings. After that, you can apply the same parameters before starting a compression sequence by using SCSetInfo. If you provide settings before calling SCCompressSequenceBegin, the dialog won't be displayed; otherwise it will be. See Inside Macintosh: QuickTime Components, page 3-8 and pages 3-15 through 3-25, for details about the format of the settings.

Also, as you may know, you can generate default parameters that also avoid the dialog by using the SCDefaultPixMapSettings, SCDefaultPictHandleSettings, and SCDefaultPictFileSettings routines. But these do require an image. This way you can avoid displaying the dialog for the first sequence, and still generate valid settings. See Inside Macintosh: QuickTime Components, pages 3-26 through 3-28, for more information about these routines.

CHAPTER 4 - IMAGE COMPRESSOR COMPONENTS

New Image Codec Functions and Flags

The interface for image codecs has three new functions and one new flag defined.

codecConditionFirstScreen is a new codec condition flag. This flag is set when the codec is decompressing an image to the first of multiple screens. In other words, if the decompressed image crosses multiple screens, then the codec can look at this flag to determine if this is the first time an image is being decompressed for each of the screens it is being decompressed to. A codec which depends on the maskBits field of the decompressParams being a valid regionHandle on CDPreDecompress (for example to do rectangular clipping, instead of bitMask clipping) needs to know that in this case it is not able to do clipping since the region handle is only passed in for the first of the screens, and the clipping would be incorrect for the subsequent screen for that image.

```
       #define codecConditionFirstScreen   (1L<<12)
```

The Standard Compression dialog box now provides Compressor components the option of displaying their own settings within the dialog box. If a compressor supports the dialog, an additional button will appear. The compressor's settings are saved with the standard compressor settings when the SCGetInfo call is used with the scCodecSettingsType flag. The codec can implement the functionality using the following three routines.

CDRequestSettings

CDRequestSettings allows the display of a dialog box of additional compression settings specific to the codec. This information is stored in a settings handle. The codec can store whatever data in any format it wants in the settings handle and resize it accordingly. It should store some type of tag or version information that it can use to verify that the data belongs to the codec. The codec should not dispose of the handle.

```
pascal ComponentResult CDRequestSettings(ComponentInstance ci, Handle settings, Rect
*rp, ModalFilterProcPtr filterProc)
```

ci	Component instance of codec.
settings	Handle of data specific to the codec. If the handle is empty, the codec should use some type of default settings.
rp	Pointer to rectangle giving the coordinates of the Standard Compression dialog box in screen coordinates. The codec can use this to position its dialog box in the same area of the screen.
filterProc	A pointer to modal dialog filter proc that the codec must either pass to ModalDialog or call at the beginning of the codec dialogs filter. This proc gives the calling application and Standard Compression a chance to process update events.

CDGetSettings

CDGetSettings allows a codec to get the settings chosen by a user. From this call, the codec should return its current internal settings. If there are no current settings or the settings are the same as the defaults, the codec can set the handle to empty.

```
pascal ComponentResult CDGetSettings(ComponentInstance ci, Handle settings)
```

ci	Component instance of codec.
settings	A handle that the codec should resize and fill in with the current internal settings. It should be resized to empty if there are no current internal settings.

CDSetSettings

CDSetSettings allows a codec to set the settings of the optional dialog box. Set the codec's current internal settings to the state specified in the settings handle. The codec should always do a validity check on the contents of the handle so that invalid settings are never used.

```
pascal ComponentResult CDSetSettings(ComponentInstance ci, Handle settings)
```

ci	Component instance of Movie Import component.
settings	A handle to internal settings originally returned by either the CDRequestSettings or CDGetSettings calls. The codec should set its internal settings to match those of the settings handle. Because the codec does not own the handle, it should not dispose of it, and should only copy its contents, not the handle itself. If the settings handle passed in is empty, the codec should set its internal settings to a default state.

Image Compression Manager Utility Functions

QuickTake Codecs and Picture Format

The new Apple QuickTake digital camera stores its pictures as compressed PICT files. This system includes a new codec that’s necessary to decompress the images, but no new file type. The QuickTake 100 Digital Camera Developer Note contains extensive information about talking to the camera from both Macintosh and Windows machines. It also documents the picture formats. The QuickTake Camera Software Development Kit is available from APDA.

Controlling Sequence Grabber Components

SGGrabPict - grabPictCurrent

The Sequence Grabber component has just one flag added to it. grabPictCurrentImage is a new flag to the SGGrabPict call. It provides the fastest possible image capture, but may fail under certain circumstances. This failure is not fatal; it just will not return a picture. You can then call SGGrabPict again without the flag set. The routine does not pause the current preview or grab the next frame. It causes the currently displayed image to be captured. It is a good idea to call SGPause yourself before calling SGGrabPict with this flag.

```
        grabPictCurrentImage = 4
```

CHAPTER 8 - VIDEO DIGITIZER COMPONENTS

Constants and Data Types

AV Systems, Video Digitizer (vdig) Specifications

Q: I'm working on a video-conferencing solution that uses the video digitizer (vdig) incorporated in the Macintosh Quadra 840av. I want to capture data from the system's built-in video hardware using the VDCompressOneFrame and VDCompressDone calls. I have the following questions about the vdig that supports the 840av built-in video hardware:

• What's the header and data format for the captured video?
• What's the compressor type (cType) for this compression format?
• Does this compressor support more than one spatial compression setting and, if so, what are the data formats for the compression settings?

___

A: We can't provide information regarding the data format of the captured video. It's considered proprietary and confidential, except in cases where the codec in use is an industry standard like JPEG. Fortunately, you don't need to know the data format if you're using the correct QuickTime vdig and Image Compression Manager calls to manipulate the data.

We don't think you should use the vdig directly, but if you do, you can call VDGetCompressionType to determine the compression types it supports. You can select the compression type you want to use by calling VDSetCompression. Since the vdig uses standard codecs for compression, you don't need to know the data format; all you have to do is use the codec to decompress the image data when you want to draw it. Call VDGetImageDescription to get an image description handle, which you can pass to DecompressImage along with a pointer to the data, and the Image Compression Manager will take care of decompressing the data as long as the correct codec is available.

We don't recommend using vdigs directly because every one is different and supports different features. They can be pretty hard to work with because your code will require a lot of error handling and workarounds. The sequence grabber was written to provide a seamless interface between any vdig and applications, so you can use the sequence grabber as the engine for your video-conferencing system. It was designed with this kind of flexibility in mind. For more information about the sequence grabber, see Chapter 6, "Sequence Grabber Channel Components," in Inside Macintosh: QuickTime Components.

Using the sequence grabber with the right flags, you can get high-performance grabs, even over the network. You do this by supplying application-defined functions to the sequence grabber component. If you replace the grab function on the receiver side, you can use the sequence grabber to grab right off the network on that end. On the sender side, you can replace the data function so that you'll be able to write the frames out over the network, using whatever network protocol you like.

Video Digitizer Component Functions

Support of Off-Screen Digitizing while Using a Video Digitizer

Q: When we try to digitize frames (grabbed with QuickTime) into an off-screen pixmap, our VDGrabOneFrame call crashes. How would you suggest we do this?

A: We recommend using the Sequence Grabber to digitize frames into an off-screen pixmap.

CHAPTER 9 - MOVIE DATA EXCHANGE COMPONENTS

New Export Component Functions

Export components have two new components, four new flags, one new error, two new functions, one new data structure, and an enhancement to the Sound Export component. The Text Movie Export component and the Audio CD Movie Import component were introduced earlier in this Note.

Four new flags were introduced for these components with QuickTime 1.6.1:

```
canMovieExportAuxDataHandle          = 128
canMovieImportValidateHandles        = 256
canMovieImportValidateFiles          = 512
dontRegisterWithEasyOpen             = 1024
```

canMovieExportAuxDataHandle is a Movie Export component flag. A Movie Export component that supports the MovieExportGetAuxiliaryData call should also now set the canMovieExportAuxDataHandle flag in its ComponentFlags.

canMovieImportValidateHandles is a Movie Import component flag. A Movie Import component should set this flag if it can import handles and wants to validate them. Validation is the process of verifying a handle and checking for corruption. If your movie import component can and wants to validate handles, then set this flag.

canMovieImportValidateFiles is a Movie Import component flag. A Movie Import component should set this flag if it can validate files and wants to validate them.

dontRegisterWithEasyOpen is a Movie Import component flag. A Movie Import component should set this flag if Macintosh Easy Open is installed and your component does not want to be registered. You set this flag if you want to handle interactions with Macintosh Easy Open yourself.

The error auxiliaryExportDataUnavailable has been added. A Movie Export component returns this when MovieExportGetAuxiliaryData is called requesting a type of auxiliary data that the component cannot generate.

```
auxiliaryExportDataUnavailable = -2058
```

The Sound Movie Export component has been updated to take advantage of the new Sound Manager. Previously, only the first sound track in the movie was exported. Now sound tracks are mixed together before being exported. If your application wants to take advantage of the sound mixing, you can use PutMovieIntoTypedHandle. It will take advantage of the Export component. Furthermore, you can now specify the format of the exported sound, so you can convert 16-bit sound to 8-bit sound, or reduce stereo to mono.

MovieExportSetSampleDescription

MovieExportSetSampleDescription allows an application to request the format of the exported data; the routine MovieExportSetSampleDescription has been added. This call is currently supported by the Sound Movie Export component.

```
pascal ComponentResult MovieExportSetSampleDescription(MovieExportComponent ci,
        SampleDescriptionHandle desc, OSType mediaType)
```

ci	Component Instance of Movie Import component.
desc	Handle to a valid QuickTime sample description.
mediaType	The type of the media that the sample description is from.

Errors:

badComponentInstance	0x80008001	Get a new component instance.

MovieImportGetAuxiliaryDataType

MovieImportGetAuxiliaryDataType returns the type of the auxiliary data that it can accept. This is useful if you are interested with import components directly. For example, if you call the Text Import component with this call, it will indicate that it can accept 'styl' information.

```
pascal ComponentResult MovieImportGetAuxiliaryDataType(MovieImportComponent ci,
        OSType *auxType)
```

ci	The Movie Import component instance. Retrieve it with OpenDefaultComponent or OpenComponent.
auxType	Pointer to the type of auxiliary data it can import. For example, a Text Import component can bring in 'text' data. But, if it says it can return 'styl', then it will import the style information as well.

Errors:

badComponentInstance	0x80008001	Your Movie Import component reference is bad.

MovieImportValidate

MovieImportValidate is a new Movie Import component routine. Validation is a method of checking and verifying data which will passed to your component. If your component can and wants to validate (see flags above), then you need to implement this call.

```
pascal ComponentResult MovieImportValidate(MovieImportComponent ci,     const FSSpec
       *theFile, Handle theData, Boolean *valid)
```

ci	The Movie Import component instance. Retrieve it with OpenDefaultComponent or OpenComponent.
theFile	The file to validate.
theData	The data to validate.
valid	Return true if the data and/or file is valid. Return false if the data and/or file.

Errors:

badComponentInstance	0x80008001	Your Movie Import component reference is bad.

TextDisplayData

TextDisplayData is a new data structure for the Text Export component. This data is useful after a text track has been exported. An application may want to know the way the text was stored as a track. You can use TextExportGetDisplayData to retrieve this data.

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

typedef ComponentInstance TextExportComponent;
```

TextExportGetDisplayData

TextExportGetDisplayData returns the text display data for the text sample that was last exported by the given Text Export component. After exporting text from a text track, it is often useful to find out about the text track characteristics. This data structure contains this extra information.

```
pascal ComponentResult     TextExportGetDisplayData(TextExportComponent ci,
                           TextDisplayData *textDisplay)

ci                         The Text Export component instance. Retrieve it with
                           OpenDefaultComponent or OpenComponent.
textDisplay                Pointer to the text display data.
```

Errors:

```
badComponentInstance  0x80008001    Your Text Export component instance is bad.
```

The style information is obtained by calling MovieExportGetAuxiliaryData on the Text Export component instance.

CHAPTER 12 - PREVIEW COMPONENTS

Functions

NIM QuickTime Components Errata: Display Image Data as Preview Bug

The code snippet that shows how to display image data as a preview has a problem concerning how to release picture resources (page 12-9):

```
bail:
       if (resRef) CloseResFile(resRef);
       if (thePict) DisposHandle(thePict);
       UseResFile(saveRes);
       return err;
```

It should be:

```
bail:
       if (resRef) CloseResFile(resRef);
       UseResFile(saveRes);
       return err;
```

The extra DisposHandle may crash the system since the CloseResFile automatically releases the Picthandle.

Further Reference:

- *Inside Macintosh*, QuickTime
- *Inside Macintosh*, QuickTime Components
- QuickTime 2.0 SDK Documentation
- QT 1 - Inside Macintosh:QuickTime Tech Note
- QT 3 - QuickTime for Windows Tech Note