“See Movie Run”

aka

Making Your Application QuickTime™ Literate

by Joe Zobkiw

Internet: zobkiw@world.std.com
America Online: AFL Zobkiw

Introduction

This paper discusses QuickTime and how you can easily implement some of its features in your application. QuickTime allows you numerous options and quite a bit of flexibility and extendibility. Anything you think you might want to do to still pictures or motion video you can accomplish via QuickTime. To make your application QuickTime *literate* you must support two or more of the following QuickTime features:

- Playback of movies (with the standard controller).
- Still image compression.
- Saving of data as QuickTime movie.
- Standard preview dialog box.
- Cut, copy, and paste of movie data.

This paper focuses on the playback of movies using the standard movie controller and the standard preview dialog box. This is what most application developers will want to implement first. These features allow you to import a movie file and play it. The file can contain video, audio, or both. An example THINK C project using the THINK Class Library is included to demonstrate this.

If you are not familiar with QuickTime or the Movie ToolBox you should obtain the QuickTime Developers Kit from the Apple Programmers and Developers Association (APDA). This paper assumes the reader has knowledge of the Macintosh ToolBox and basic knowledge of QuickTime, the Movie ToolBox, and the standard movie controller.

Quick Overview

As mentioned above, QuickTime is not just movies, it is a group of three ToolBox Managers that work together to give your applications access to movie playing (via the Movie ToolBox), image compression (via the Image Compression Manager) and other components such as the standard movie controller (via the Component Manager). QuickTime allows you to implement your own compressors and other components for custom applications. Once implemented, other developers can also use your components.

This paper focuses on use of the Movie ToolBox and Component Manager to play movies via the standard movie controller. This is all that many applications will need to be able to do. Editing of movies can be left to other applications like Adobe Premiere and DiVa’s VideoShop -- two excellent movie creation products.

Quick Implementation

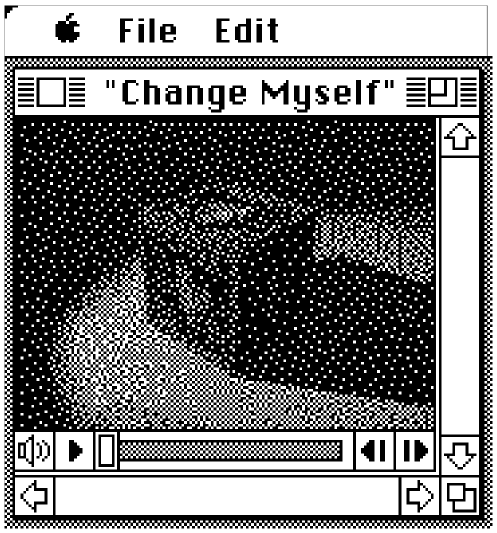
When your application uses QuickTime there are a few routines you must call before you can call any of the QuickTime specific routines. When your application starts up you must use `Gestalt` to determine if QuickTime is available on the current configuration. The following routine returns true if QuickTime is installed and false otherwise. If QuickTime is available, version also contains the version of QuickTime. This routine is contained in the file QuickTime Utilities.c and is explained below.

```
Boolean  QuickTimeIsInstalled(long *version)
```

Once you have determined that QuickTime is available you must call `EnterMovies` to alert QuickTime that your application will be taking advantage of its features. When `EnterMovies` is called, QuickTime allocates special data structures for your application that it uses while it is running. You can now use the Movie ToolBox to play, edit, and otherwise manipulate movies. When your application quits, you should call `ExitMovies` so QuickTime can deallocate its private storage for your application.

See Movie Run

The THINK Class Library implementation for this paper is relatively straightforward. The main QuickTime class is the CMovie class. This class implements a movie object that handles displaying and playing of a QuickTime movie. CMovie is a subclass of CPane so it is easy to utilize as a drawing environment. The CMovie class automatically adds the standard movie controller to the movie it is currently using. The standard movie controller gives you a simple way to start, stop, and "walk the frames" of a movie.



CMovie pane with standard movie controller

CMovie uses only a few QuickTime calls to perform its magic. It centers around a few Movie ToolBox calls for loading the movie resources into RAM such as `OpenMovieFile` and `NewMovieFromFile`. A few utility-type routines such as `GetMovieBox` and `GoToBeginningOfMovie` are used to prepare the movie for playback. Lastly, we use `GetMoviesError` to check for errors from any of our Movie ToolBox calls that do not return an error code as a function result.

To create the standard movie controller and attach it to the movie we use the Component Manager call `NewMovieController`. This handles the majority of the dirty work of creating the controller and attaching it the movie. There are other ways to accomplish this but this is by far the easiest since it sizes the controller, etc.

The standard movie controller needs to be passed ToolBox events received from `WaitNextEvent` in order for it to update itself, handle user actions, etc. We use the Component Manager routine `IsMCPlayerEvent` to pass the events to the controller. Functionality like this does not come naturally to a subclass of CPane in the THINK Class Library so a bit of trickery is involved which is explained in more detail below.

Lastly, we need to make sure that a playing movie has enough time to update itself. For this we summon the `MoviesTask` function call during our CPane’s `Dawdle` method.

This routine gives the Movie ToolBox time to play the movie and update it on the screen, even in the background!

<u>Using CMovie</u>

To use a CMovie object within your application, you first have to make sure you create and initialize the gEBCollaborator and the new CEBSwitchboard objects.

Your application class should also create the gEBCollaborator object within its initialization method and dispose of it in its `Exit` method.

```
void  CQTApp::IQTApp(void)
{
      OSErr err = noErr;

      CApplication::IApplication(kExtraMasters, kRainyDayFund,
                                 kCriticalBalance, kToolboxBalance);

      if (QuickTimeIsInstalled(&gQuickTimeVersion) == true) {
            FailOSErr(EnterMovies());
      } else {
            gQuickTimeVersion = 0;
            FailOSErr(gestaltUndefSelectorErr);
      }

      gEBCollaborator = new(CEBCollaborator);
      gEBCollaborator->IEBCollaborator();
}
```

Your application class should override the `MakeSwitchboard` method and replace it with the following method to create a CEBSwitchboard switchboard:

```
void  CQTApp::MakeSwitchboard(void)
{
      itsSwitchboard = new(CEBSwitchboard);
      ((CEBSwitchboard*)itsSwitchboard)->IEBSwitchboard();
}
```

Once these few things are taken care of we can begin to create and use CMovie objects. CMovies initialization method looks exactly like that of CPane except for one small change. The last parameter to be passed to CMovie is a pointer to an FSSpec record that is either nil or contains a valid movie file description.

```
void  CMovie::IMovie(
      CView             *anEnclosure,
      CBureaucrat       *aSupervisor,
      short             aWidth,
      short               aHeight,
      short               aHEncl,
      short               aVEncl,
      SizingOption      aHSizing,
      SizingOption      aVSizing,
      FSSpec            *movieSpec)
```

CMovie will use this movieSpec to either open and prepare the movie for playback or if it is nil will paint itself gray.

The only other routine you will really need to worry about using (unless you are a real hacker) is the `ImportMovie` method. You pass `ImportMovie` a pointer to an FSSpec record that contains a valid movie file description. `ImportMovie` will replace the current movie with the one described in the spec parameter. If spec is nil it is ignored.

```
void  CMovie::ImportMovie(FSSpec *spec)
```

To experiment with the CMovie class, you may want to simply edit the See Movie Run application. Even if you don’t want to edit the existing source code, looking it over will undoubtedly shed some light on the way this class works and how it can be expanded. It is a very simple implementation and does not take advantage of advanced data hiding and other features of object oriented programming. These things could be implemented quite easily if you need them.

Controller Trickery

Because the standard movie controller needs access to events and because it is part of a CMovie (which normally doesn’t receive every event) we needed some sort of mechanism to pass ToolBox events to the controller.

At application launch we allocate a global “event broadcasting” Collaborator (CEBCollaborator class) instance known as gEBCollaborator. This collaborator is responsible for broadcasting events returned from `WaitNextEvent` to any of its dependents via its unique BroadcastEvent method.

When a CMovie object is created, one of the first things it does is to register itself with the gEBCollaborator as an entity that needs access to ToolBox events. It does this by sending gEBCollaborator a `DependUpon` message.

We also subclass CSwitchboard (by creating a CEBSwitchboard class) and override its `DispatchEvent` and `DoIdle` messages. When our gEBSwitchboard receives an event for processing, it first sends the event within a `BroadcastEvent` message to the gEBCollaborator.

```
void  CEBSwitchboard::DispatchEvent(EventRecord *macEvent)
{
      gEBCollaborator->BroadcastEvent(macEvent);
      inherited::DispatchEvent(macEvent);
}


void  CEBSwitchboard::DoIdle(EventRecord *macEvent)
{
      gEBCollaborator->BroadcastEvent(macEvent);
      inherited::DoIdle(macEvent);
}
```

gEBCollaborator continues by sending a `BroadcastChange` message to its superclass that ultimately calls the movie object, one of its dependents, by sending it a `ProviderChanged` message.

```
void  CEBCollaborator::BroadcastEvent(EventRecord *macEvent)
{
      this->BroadcastChange(kEventRecordReason, (Ptr)macEvent);
}
```

The movie object can then parse the `ProviderChanged` message and extract the event from it to pass to its controller via the QuickTime `MCIsPlayerEvent` function. If the controller handles the event, CMovie changes it to a nullEvent so it is effectively ignored elsewhere, otherwise, it passes through unscathed to the next object in line.

This scheme can be used in any situation that requires objects to receive events before the application gets a chance to process them and works out very well. Special thanks to Chris Wysocki for assisting in thinking this one through!

QuickTime Utilities.c

The file QuickTime Utilities.c contains a few useful routines when dealing with the CMovie class and QuickTime movies in general.

```
Boolean GetMovieFileFSSpec(FSSpec *aFileSpec)
```

Shows the standard movie preview dialog box and returns true and a valid FSSpec for the movie file chosen if the user pressed Open. If the user canceled aFileSpec is undefined and `GetMovieFileFSSpec` returns false. This routine is called when the user selects Open… or Import Movie… from the File menu.

```
Boolean QuickTimeIsInstalled(long *version)
```

Returns the current version of QuickTime in version and true if QuickTime is installed. If QuickTime is not installed, version will be zero and `QuickTimeIsInstalled` returns false.

What Else Is Possible?

See Movie Run simply plays movies. The application does not perform any editing of the movie data, resizing of the movie pane, dragging of the movie pane, etc. You might also choose to draw the CMovie pane into an offscreen GWorld or PixMap and blit it to the screen all at once using `CopyBits` to avoid flicker during update events. You could even add odd-shaped “regioned” movies. The possibilities are endless!

These features are not too difficult to add and may be implemented in the future. In the meantime, if anyone changes the source, please send me a copy.

In Conclusion

The best way to learn about QuickTime is to dive in! I will be the first to admit that the online documentation on the QuickTime CD is very difficult to read on any Macintosh with less than a full page display. I suggest printing as much of the documentation as you can. As I’ve learned from this excursion into QuickTime you can get by at first if you only print the introduction, Movie ToolBox, and the standard controller section of the Component Manager. You may also want to skip the Matrix discussion in the Movie ToolBox sections unless you are into masochistic routines.

With a little work (See Movie Run was implemented and debugged in about 20 man-hours -- including reading about QuickTime!) you can implement QuickTime capabilities into your application and make it “cool.” QuickTime is paving the road to the future, why not follow it?

Special thanks to Neil Ticktin and everyone at MacTutor Magazine, MacWorld Magazine for having the Expo in Boston, America Online and Software Tool & Die for providing electronic access to my friends and co-workers, Reneé Lockett for supporting my programming late into the night when she was trying to sleep, Chris Wysocki for the event broadcasting idea, Brian Gaines for the great suggestions and for setting me straight on a few QuickTime and TCL gotchas, and everyone who proofread and/or gave suggestions to make this paper and source code what it is today. Also, a special thanks to Ralph Bellofatto who gave me my first break and then convinced me that I had the talent to succeed as a Macintosh software engineer.