The QuickTime XCMDs

Ken Doyle
QuickTime Software Group
Apple Computer, Inc.
April 26, 1994

I am not an author (nor do I play one on TV).

The QuickTime XCMDs are a small set of XCMDs that allow HyperCard users access to many of the features of the QuickTime® library of software. The QTMovie XCMD can be used to play QuickTime movies either in a window or directly onto the screen. The QTRecordMovie XCMD displays a window in which video coming from a digitizer card can be viewed. You can then send commands to the window to capture and create your own movies or picture files. The QTEditMovie XCMD allows you to perform various editing functions including cut, copy, and paste of individual tracks among different movies, adding text tracks to a movie, setting a clip region on a movie, and many other editing functions. The QTPict XCMD performs a variety of still picture related utilities including displaying a picture on a card, compressing pictures, and allowing control over the clipping region of the card window.

It is recommended that you give HyperCard at least a two megabyte partition when using the QuickTime XCMDs.

The QTMovie XCMD: Getting Started

The QTMovie XCMD allows you to play movies in a window in HyperCard, using HyperCard 2.0's XWindow facility. An alternative method called *Direct* movies does not play into a separate window but rather plays directly into the card window's port. This method has some advantages and disadvantages, but in particular allows one to use the XCMD in SuperCard and Macromind Director. Direct Movies are discussed later.

Window Movies

Playing a movie in a window is as simple as sending the following command (either from the message box or from a HyperTalk script):

```
QTMovie  OpenMovie, Document, "MyDisk:MyMovie", "20,20"
```

A document type window will appear with "MyMovie" playing at location 20,20 within the card window. You can then use all the features of the movie controller that appears at the bottom of the window to navigate through the movie. When you are done you can close the window by clicking on the close box. That's all you really need to know to play a movie. Naturally there are a few additional options and the next 743 pages attempts to explain them.

The Basics

The basic form of the QTMovie command is:

    QTMovie OpenMovie, windowType, <fileName>, location [,options...]

The first parameter is "OpenMovie". This tells the XCMD that we want to open a new movie window. The second parameter is the windowType. The window type can be one of the following: Document, Windoid, TallWindoid, Plain, Dialog, AltDialog, or Borderless (see illustration below). Other options for this parameter are discussed in the Advanced Topics section. The third parameter is the name of a movie file. If the movie is in the same folder as the stack, you need only name the file, otherwise, the full path name must be provided. The location parameter can be one of a few different options. You can specify a point or a rectangle for the location parameter. If a point is specified, then the movie is shown at its normal size at the point specified. The point is in the local coordinates of the card window. If a rectangle is specified, then the movie is scaled to fit the rectangle. The top left of the rectangle specifies the location. Again, the rectangle is in local coordinates. Alternatively, you can specify one of the following literals for the location parameter: *card*, *largest*, *deepest*, or *main*. These will cause the movie to be centered on the same screen as the card window, the screen with the largest area, the screen with the greatest bit depth, or the main screen, respectively.

You can specify a list of optional parameters in any order after the location parameter. The most common options are:

    mute   - start the movie in a muted state
    paused   - start the movie in a paused state at time zero
    loop   - when play hits the end, loop back to the beginning
    invisible - initially, do not show the window (or movie if direct)
    noController   - do not display a movie controller in the window

The above options are used to override the default behavior of the specific features. Thus the default is for a movie to be shown playing with sound on. The window will have a controller and it will be visible. Additional advanced options are discussed later.

After calling QTMovie OpenMovie, the HyperTalk global "the result" will contain an error message if some problem was encountered in attempting to open the movie. The first word of the error message is always the word "Error". Thus when calling QTMovie, you should always follow up with an error check:

    QTMovie OpenMovie, Document, "MyDisk:MyMovie", "20,20"
    get the result
    if "Error" is in it then <do error handling>

As mentioned above the windowType parameter specifies the window type in which to display the movie. The window types are:

document   windoid   tallWindoid

plain   dialog altDialog   borderless

All of the window examples above are shown without a controller.

Examples of opening a new movie window:

    QTMovie OpenMovie, Windoid, "HD:Movies:Running Horses", "100,100"
    -- plays a movie in a windoid located at 100,100 with a movie controller

    QTMovie OpenMovie, Plain, cd field fileName, the rect of btn movie, nocontroller
    -- plays a movie (whose name is obtained form the card field "fileName") in a Plain window scaled to the size and
    location of button "movie". The movie controller is not visible.

    QTMovie OpenMovie, Document, userReply, card, paused, invisible
    -- starts a movie in an hidden document window, paused at the beginning of the movie. The name of the movie
    is contained in a the script variable "userReply", which could have been the result of a HyperTalk "answer file"
    dialog. The window is made visible by executing show window <windowName> at which point it will appear
    centered in the card window's screen.

Controlling Movies

Unless you specified *noController* with a window movie, you will get the QuickTime standard movie
controller in the window. It will always be at the bottom of the window.

With this controller, you can use the slider to quickly position yourself anywhere in the movie. You can also use the step forward and reverse buttons to fine tune your position. The play/pause button toggles between play (a black triangle) and pause (two vertical lines). The speaker icon on the left controls the movie volume. In addition, you can click in the movie itself to pause a movie and double click to start it. This is handy when you do not have the controller displayed.

You can also control movies by sending messages to the window from HyperTalk. For a window movie the general form of a window message is:

> send <message> to window <windowName>

The window name defaults to the file name of the movie with the volume path extracted. (You can use the windowName property to change the window name.)
For example, to have the XCMD play the movie in reverse (a function not intuitively available in the movie controller) you would send the message:

> send Reverse to window "Juice Bottles"

There are a number of messages you can send to the movie window. The most common of those are:

> Play   - Play Forward (sets speed to last rate set (see rate property below))
> Pause   - Pauses the movie
> Reverse - Play Reverse (negate the last rate set, and play at that speed)
> StepFwd- Step Forward 1 "frame"
> StepRev - Step Backward 1 "frame"
> HideController   - Hide the play controller
> ShowController   - Show the play controller

The control commands that you can send are: Play, Reverse, Pause, StepFwd, and StepRev. Play will play the movie at whatever its current rate setting is. The default rate is whatever preferred rate is saved with the movie (normally 1). You change the current rate setting by setting the rate property (see below). Pause will halt the movie but its current rate setting will be retained. Reverse negates the current rate setting and then tells the movie to play. If the rate was already negative then Reverse will cause the movie to play in the forward direction. Again, more advanced messages are presented in the Advanced Topics section.

Getting and Setting Properties of Movies

Another means of controlling movies is by setting various properties of the window. The difference between this and sending a message is that there is a value associated with a property. The general form of the calls for window movies is:

```
set <property> of window <windowName> to <value>
get <property> of window <windowName>
```

There are some properties that you can get but not set, such as the duration property, which returns the duration of the movie (in the movie's time scale). Most of the settable properties you can get (in many cases it just returns whatever you set before.) All together there are over seventy properties associated with the QTMovie XCMD, ranging from the common and useful to the rare and obscure. The common and useful I will discuss now; the rare and obscure (and those in between) I'll save for the Advanced Topics section (which I really do intend to write).

The common properties related to the movie itself are:

```
currTime- get or set (in the movie's time scale) the current time (does not pause)
duration  - return (in the movie's time scale) the duration of the movie
movieScale       - return the time base of the movie (time units per second)
rate      - fixed number - sets playback rate of the movie (negative for playing in
                    reverse). If movie was paused, it stays paused. Use the play command
                    to start movie going at new rate.
loop      - turns on or off loop mode (set to true or false)
audioLevel       - the current audio level (0-256, or higher with QT1.6 and new Sound Mgr)
mute      - turns on or off muting (set to true or false)
```

Four properties affect the size or location of the movie or window (these are discussed in greater detail under Window and Movie Positioning and Resizing):

```
windowLoc        - a new location for the window
windowRect       - a new rect for the window
movieLoc         - change the location of the movie within its window
movieRect        - scale the movie into a new rectangle
```

Example calls:

```
put currTime of window "High Jumper" into field saveTime
get audioLevel of window "Tiger"
put movieLoc of window 'Race Car" into field raceCarLoc
set currTime of window "High Jumper" to field saveTime
set rate of window "Race Car" to 2.5
put duration of window currWindow / movieScale of window currWindow into movieSeconds
```

Closing the Window

To close a window movie you can click in the window's close box or you can use the close window command:

```
close window "Running Horses"
```

The movie's data structures are automatically freed when the window is closed.

Direct Movies

Direct movies play directly into the card window. You indicate that you want a direct movie by specifying Direct for the windowType. Assuming there is no error, "the result" will contain a movie id that you will need to save for subsequent control of the movie.

```
global movieID
QTMovie OpenMovie, Direct, field fileName, topLeft of btn movieLoc
get the result
if "Error" is in it then <do error handling>
 else put it into movieID
```

For window movies, HyperCard automatically gives the window idle time and hence movies keep running without any need for action on your part. With Direct movies, however, the automatic idle facility does not exist. You need to give the movie idle time by installing an Idle handler in your card (or background or stack) script. The Idle handler looks like this:

```
on idle
    global movieID
    if movieID is not empty then
        QTMovie  Direct, movieID, Idle
    end if
end idle
```

Notice that in this case the Direct keyword is the first parameter to QTMovie. This is in fact the general form of all messages sent to direct movies:

```
QTMovie Direct, movieID, <message> [,options...]
```

The second parameter is always the movie id that was returned by the OpenMovie command. The third parameter can be one of the messages listed above for window movies (such as play or pause).

```
 QTMovie Direct, movieID, Play
 QTMovie Direct, movieID, StepFwd
```

To get or set properties the third parameter is Get or Set, in which case there is a fourth property which is the name of the property. For Set there is a fifth parameter, the value to which to set the property. For Get, the property value is returned in "the result".

```
QTMovie Direct, movieID, Get, currTime
put the result into field saveTime
QTMovie Direct, movieID, Set, rate, -3.0
QTMovie Direct, movieID, Set, currTime, field saveTime
```

When you are done with a Direct movie, you need to dispose of it so that the movie's data structures are released. This is done with the Dispose message:

```
 global movieID
 QTMovie Direct, movieID, Dispose
 put empty into movieID
```

Since Direct movies are played directly onto the card window, the display is volatile. This means that if HyperCard needs to update the card, the movie display will be erased. This is the main disadvantage of using Direct movies. You need to be aware of this and try to avoid unnecessary updating. The following script will refresh a Direct movie:

```
on refresh
    global movieID

    QTMovie Direct, movieID, Get, movieLoc
    put the result into xy
    QTMovie Direct, movieID, Set, movieLoc, xy
end refresh
```

This might be a handy script to call from the moveWindow handler, since the movie could get erased if the card window is dragged. You might also explore the use of the clipping commands in the QTPict XCMD to try to protect the movie from being erased.

There two main reasons for direct movies' existence. One is the availability of a set of special features that are only available with direct movies. The other is the ability to play direct movies in SuperCard and Macromind Director. Since SuperCard and Director do not support the the HyperCard XWindow interface, direct movies are the only way to use QTMovie in those applications. Indeed, there are other ways to play QuickTime movies in those applications, but if for some warped reason you want to use QTMovie, direct movies are the way to go.

There are several other features of Direct movies that will be covered in the Advanced Topics section.

QTMovie: Advanced Topics

Included in this section is a discussion of the remaining OpenMovie options and window properties as well as some advanced Direct movie features.

Call Back Messages

Call Back messages are messages that are sent from the XCMD back to HyperCard. It requires the XWindow interface so will not will for direct movies (except for the timed call back message). You can set up a number of different call backs that the XCMD will perform. You name the handler within your card, background, or stack script that is to be called when certain conditions occur by setting the appropriate message name property.

If you set the **mouseDownMsg** property, your handler will be called whenever the mouse is clicked in the movie area of the window (not when it is clicked in the controller or the title bar). The mouse click location (in the coordinates of the movie window) is sent as the first parameter to the mouseDownMsg. The current time in the movie is passed as the second parameter, and the name of the movie window is in the third parameter.

The **timedCallBack** property is a bit different. It specifies both a handler and a time (in movie time) when the handler should be called. The handler and the time are separated by a space. (This is handled quite easily by using "&&" in HyperTalk.) If the movie is playing in the forward direction, the message

is called when the movie time is greater than or equal to the call back time; if it is playing in reverse then the message is called when movie time is less than or equal to the call back time. The timed call back message is called with the window name as its parameter. The time is expressed in movie time, a value which can be obtained by getting the currTime property of the window. You can also specify "end" for the call back time, in which case the call back is made when the movie hits the end. Unlike the other call back messages, callBackMsg is cleared whenever it is executed. You can reset the timedCallBack from within the callback handler if you wish.

The **windowCloseMsg** is called when the window is about to be closed (for example when the user clicks in the close box). It is called with the window name as a parameter.

The **cursorMsg** is called repeatedly whenever the cursor is over movie area of the window. The parameters passed are the mouse location (again in the coordinates of the movie window) and the window name. The intent of this callback is to allow the user to set the cursor shape when the cursor is over the window.

The **movieControlMsg** is called when the movie controller receives a play, go to time, or set sound level command. Three parameters are passed to the handler: an action code, an action parameter, and the window name. Three action codes are currently defined: play is 8, go to time is 12, and set sound level is 14. (These are the codes used by the movie controller itself.) The action parameter depends on the code. For play it is the play rate (rate = 0 is pause). For go to time it is the destination time in the movie's time scale. For set sound level it is the sound level (0-255). A negative sound level indicates muting.

The **statusMsg** is called if an unexpected error occurs during movie playback. The parameters passed are an error number and the window name.

Examples:

```
set windowCloseMsg of window "Race Car" to "MovieWindowClosing"
set timedCallBack of window "Tiger" to "showMyPict" && savedTime
set cursorMsg of window "Tiger" to "CursorShape"
set movieControlMsg of window "Bozo" to "LimitSound"
```

Examples of simple handlers defined in your card script might be:

```
on MovieWindowClosing windowName
    if windowName is "Race Car" then go next card
end MovieWindowClosing

on CursorShape
    set cursor to crossHair
end CursorShape
```

Note that although CursorShape above is actually passed both a mouse location and a window name as parameters, it does not need to list them in its definition if it does not need to make use of those parameters.

The timedCallBack can also be used with Direct movies. The window parameter will be empty when the call back message is called.

```
QTMovie Direct,windowID,Set,timedCallBack,"showMyPict" && saveTime
```

The mouseDown message overrides the standard behavior of clicking in the movie to stop and start the movie. If you still want his behavior, then you can send the **passMouseDown** message from within your handler:

```
on MyMouseDown pt, movieTime, windowName
    if the commandKey is down then
        -- toggle the mute
        set mute of window windowName to not the mute of window windowName
    else
        -- otherwise let the click pass through
        send PassMouseDown to window windowName
    end if
end MyMouseDown
```

The opposite behavior is true of the movieControlMsg message. The movie control command that is being intercepted will be executed unless the **cancelMessage** message is called from your handler.

```
on LimitSound actionCode,param,wName
    if actionCode = 14 then
        if param > 128 then send cancelMessage to window wName
    end if
end LimitSound
```

More on Controlling Movies

When QuickTime is playing a movie it uses the movie's current rate setting to determine how fast to play the movie. Sometimes this may mean that in order to maintain that rate, some movie frames will have to be dropped (not displayed) since the decompression and display of the previous frame took too long. For some movies it is more important to see all of the frames rather than maintain any particular movie rate. For these movies you can set the **seeAllFrames** initial option (or after the window is opened, set the **seeAllFrames** property to true). In this mode no frames are dropped. They are simply displayed sequentially as fast as possible. Note, however, that sound tracks will *not* play in this mode.

If you set the **enableKeys** property to true, you can use the standard movie controller's keyboard equivalents to control the movie. Full details can be found in the standard movie controller documentation (in other words, I don't know all the keyboard shortcuts). Note that the movie window must be the "active" window for keyboard commands be sent to the window. Clicking in the window will make it active.

In addition to the loop property mentioned before you can also set the **palindrome** property (or have it as an initial option). When in palindrome mode, the movie loops back and forth between the beginning to the end of the movie (or segment, see following).

You can specify a segment of the movie to play by setting the **segmentStart** and **segmentEnd** properties:

```
set segmentStart of window "myMovie" to 100
set segmentEnd of window "myMovie" to 400
set segmentPlay of window "myMovie" to true
```

Setting **segmentPlay** to true will cause the segment specified to be played. The times specified are in movie time. If loop or palindrome is set, then the segment will be played continuously in that mode, otherwise it will stop at the end time specified. If the end time precedes the start time, then the segment will be played backwards. To exit segment play mode, set segmentPlay to false. You can have the segment loaded into RAM (as much as memory allows) by sending the **loadSegIntoRAM** message:

```
        send loadSegIntoRAM to window "myMovie"
```

You can also specify **loadMovieIntoRAM** as an option to OpenMovie. The XCMD will attempt to preload as much as the movie as possible into RAM before playing the movie.

Many QuickTime movies have video tracks whose video compression scheme uses frame differences with periodic key frames. To quickly go to the next or previous key frame in a movie you can send the **GoNextKeyFrame** or **GoPrevKeyFrame** messages.

Previously mentioned were the **ShowController** and **HideController** messages that will show or hide the movie controller. You can find out if the controller is currently visible by getting the **hasController** property, which returns true or false. Another method of showing the controller is by using the standard movie controller badge option. If you set the **badge** property to true (or set the **badge** initial option), then, whenever the controller is not visible a small movie icon (called, guess what?, a badge!) appears in the lower left portion of the movie when the movie is paused. If you click on the badge the movie controller appears. Unfortunately there is no standard interface for making the controller disappear and the badge reappear. One possible scenario is to have a timed call back message that triggers when the movie hits the end which hides the controller. The **showPoster** message will position the movie at the poster frame for the movie. You can also specify **showPoster** as an option to OpenMovie. (The QTEditMovie XCMD allows you to set the movie poster.)

In the section above on Direct movies, the **Idle** message needed to be called as frequently as possible in order for the movie to play. This is not necessary for a window movie, since HyperCard automatically gives the window idle time. However, if you have a script that contains a loop that does not exit for some time   you should send the Idle message to the movie window from within the loop.
For example, the following button script sets the rate of the movie based on the horizontal mouse position within the button. If the Idle message is not sent within the repeat loop, the movie will come to a halt while the mouse button is held down.

```
        on mouseDown
           global currWindow
           if there is not a window currWindow then exit mouseDown
           repeat while the mouse is down
              if the mouseLoc is within the rect of me then
                 put the mouseH-the left of me into dx
                 put dx / the width of me * 2 into newRate -- set rate between 0 and 2
                 set rate of window currWindow to newRate
              end if
              send idle to window currWindow
           end repeat
        end mouseDown
```

Under some circumstances, the amount of overhead used by HyperCard (or other host application) may cause movies to not get enough idle time and thus degrade play back performance. You can specify the **fastIdle** initial option (or set the **fastIdle** property to true) and get improved performance. If you specify

the fastIdle option at OpenMovie time, then when idle is called the XCMD will go into a tight loop keeping the movie going and will not return until an OSEvent occurs. This causes movies to perform better, but nothing else can go on at the same time. The cursor will not change shape. Only one movie will run at a time when fastIdle is active. The XCMD does not go into a tight loop when the movie is paused or another application is brought to the front. Note that since the tight loop is interrupted by an OSEvent, you can still click on buttons, etc. FastIdle is available for both Direct and Window movies.

Window and Movie Positioning and Resizing

There are a couple of additional initial options that have an effect on the window's size. The **clipTo** option causes the movie to be clipped to the rectangle parameter that must follow the clipTo keyword. If the **showGrowBox** option is specified then the movie controller will have a grow box with which the user can resize the window at will.

The initial window position is determined by the location parameter of the QTMovie OpenMovie call. If a point or one of the positioning keywords (deepest, main, etc) is specified, then the size of the window will be the default size of the movie (plus the movie controller if it is visible). If a rect is specified then the movie will be scaled to fit into the specified rectangle and the window will be sized accordingly. The one exception to this is if the **clipTo** option is included as one of the parameters. In this case the movie is still sized according to the position parameter as before, but the window size and position will be that of the rectangle specified after the clipTo parameter. As an example, if you had two buttons on your card – a small button centered inside of a large button – and you executed the following:

    QTMovie OpenMovie, plain, fileName, the rect of btn large, clipTo, the rect of btn small,
                                                                    nocontroller

you would get a plain window the size and location of the small button in which the center of the movie (whose size and location is that of the large button) is visible. Similarly, if you specified Direct instead of a window type in the above statement, the center portion of the movie would play at the size and location of the smaller button. The coordinate system of both the position parameter and the clipTo option is that of the current card window.

You can subsequently change the window size and location as well as the size and location of the movie within its window, or, in the case of direct movies, the size and location within the card window. To change the window size or location set the **windowRect** or **windowLoc** properties. The size of the movie will not be affected by either of these, but it will maintain its relative location within the window. The rect or point used for these properties is again in the coordinates of the card window. If you want to change the location or size of the movie within the window, you set the **movieRect** and **movieLoc** properties. These do not affect the window location or size. For example, if you doubled the size of the movieRect, the visual effect would be that of zooming into the movie. If you changed the movieLoc, the effect would be that of scrolling or panning the movie. The coordinate system differs for these properties depending on whether the movie is a window movie or a direct movie. For a window movie, movieLoc and movieRect are expressed in the coordinates of the window, thus to reset the top left corner of the movie to be at the top left of the window, you would set movieLoc to be "0,0". For direct movies, movieLoc and movieRect are in the coordinates of the card window (which is effectively the movie's window so there really isn't any difference, I suppose). Note that the movieLoc of a window movie is always initially 0,0 unless the clipTo parameter is specified, in which case the movieLoc is the difference between the

position location and the clip rect location.

The **clipRect** property can be set on a movie. This is useful mainly for direct movies, since window movies can achieve the same effect by setting the movieLoc to a negative value and setting the window size appropriately. The clipRect property specifies a portion of the movie to be displayed in the window. Its coordinates are expressed in the coordinates of the movie's window (the card window for direct movies). Its effect for direct movies is the same as the clipTo initial option.

The **eraseOnMove** property is provided for use with direct movies. It defaults to true, which means that whenever a property is set that causes the displayed rectangle to move to a different location or size in the card window, the previous location will be erased. If this property is set to false, then it will not be erased.

If you specify **showGrowBox** when you call OpenMovie, a grow box will appear in the movie controller. You can resize the window by clicking and dragging the grow box. When showGrowBox has been specified, the movie will always scale to fit in the window. This behavior overrides the description above for the various window and movie positioning commands. In particular, the clipTo option during OpenMovie is ignored if showGrowBox is also specified. In addition the movieLoc property is not settable. The movieRect and windowRect properties will resize both the movie and the window appropriately. As a result, script control over the movie position is far left flexible in this mode.

More Window Properties and Options

The name of the movie window defaults to the name of the movie file. You can change the name of the movie window by setting the **windowName** property.

The movie window can be hidden without actually closing by using the HyperTalk **Hide** command or by setting the **visible** property to false. Likewise, you can show a hidden window by using the **Show** command or by setting the visible property to true:
```
hide window currWindow
set visible of window currWindow to false
show window currWindow
set visible of window currWindow to true
```

The **closeOnFinish** initial option is provided so that you can open a movie, have it play through once, and then close by itself. You can also set the **closeOnFinish** property after the window is opened to have the same effect. When the end of the movie is reached, the window automatically closes. Note that if you set the windowCloseMsg property (see above) then you handler will be called before the window is closed.

By default, when a window is shown, the window border is drawn and then the contents are erased to white before the actual window contents are displayed. If the window is being displayed on a non-white background the effect can be undesirable. If you set the **dontPaintWhite** initial option then the erase will not occur whenever the window is shown. Alternatively you can set the **dontPaintWhite** property (to true or false) at any time after the window is opened.

In a similar vain, when a movie window is closed, the card window behind it is told to refresh the area where the closed window used to be. A common situation is to have a closeCard handler close the current movie window. This causes the card window to update, which is not really necessary in this case

since we are in the process of going to the next card. Setting the **dontInvalOnClose** property to true causes the card window to not update when the movie window is closed.

Normally window movies show up in HyperCard's palette layer so that any movie window is always in front of the card window. You can override this by specifying **documentLayer** as an option to OpenMovie. The movie window will then be part of the document layer such that when you click in the card window the movie window will go behind it. Any movie window in the palette layer will always be in front of any movie window in the document layer. If you have a movie window in the document layer that is in front of the card window, you will need to click once in the card window to activate it before you can click on buttons on the card.

You can send the **GoToBack** and **GoToFront** messages to send a movie window behind or in front of the other windows present. The window will go to the front or back of the layer (palette or document, see preceding paragraph) to which it belongs.

Replacing Movies in a Window

There are two ways to replace the currently playing movie in a window. The first (and simpler) method is to set the **newMovieFile** property of the window:

        Set newMovieFile of window "Movie Window" to "MyDisk:MyMovie1"

This will replace the currently playing movie with the new movie specified. The previous movie will be disposed (unless it is a queued movie, see below). The current movie rate and volume will be maintained. If you had specified a rectangle rather than a point for the location when the window was originally created, then the new movie will be scaled to fit that same rectangle. If you originally specified a point, then the window will be resized (if necessary) to accommodate the new movie's default size.

Note that in the above example the window name is "Movie Window". If you are going to be replacing movies in a window it is recommended that you change the name of the window to some generic name to avoid confusion (since the window name defaults to the name of the original movie).

        QTMovie OpenMovie, windoid, "MyDisk:Dancing Bear", "100,100"
        Set windowName of window "Dancing Bear" to "Movie Window"

A somewhat more complicated way to replace movies in a window is to use the **queuedMovie** feature. This has the advantage that the movies are replaced more quickly since the file will have already been opened and data structures will have been primed for playing the movie. (Due to inadequacies in XWindow syntax, the commands to queue up, play, and delete queued movies are somewhat awkward.)

To queue up a movie to be played later you need to set up the queued movie and save a reference to it. You can set up as many queued movies as memory will allow:

        Set queuedMovie of window "Movie Window" to "MyDisk:MyMovie1"
        put the result into queuedMovie1

        Set queuedMovie of window "Movie Window" to "MyDisk:MyMovie2"
        put the result into queuedMovie2

These commands have no immediate effect on the window. To replace the current movie with a queued movie you set the **activeMovie** property:

>   Set activeMovie of window "Movie Window" to queuedMovie1

Normally when a movie is replaced by another movie, either by setting newMovieFile or activeMovie, the previous movie is disposed. However, queued movies are not disposed when they are replaced. If you need to dispose of a queued movie, you must explicitly dispose it by setting **disposeQueuedMovie** [winner of the 1993 most awkward syntax award]:

>   Set disposeQueuedMovie of window "Movie Window" to queuedMovie2

If the queuedMovie you dispose is the currently playing movie, the next queued movie is played. If there are no other queued movies, the window is closed automatically.

When you close a window with queued movies, all movies are disposed of automatically.

An example of where the queued movie feature might come in handy might be for an adventure game where you would queue up movies of adjacent rooms as you enter a new room, while disposing movies of no longer adjacent rooms.

If you set the **replaceTime** property before setting newMovieFile or activeMovie, the new movie will start at the specified time.

Advanced Window Type Options

In addition to the previously mentioned standard window types that you can use in the OpenMovie call, there are two other window type options available. You can pass an integer value that corresponds to a window type other than the standard ones offered. Note that this window type value is determined by multiplying the corresponding WDEF id by 16 and then adding any WDEF specific value between 0 and 15. For example, my System File (and possibly yours, too) has a WDEF resource whose id is 1. If I call QTMovie OpenMovie with a window type of 16, I get a round rect type window with a black title bar. Using other values between 16 and 31 has an effect on the roundness of the corners.

The other advanced option for the window type is to specify **movieWDEF**. If you use movieWDEF for the window type, the movie will appear in a window whose shape is determined by the clip region of the movie. For normal rectangular movies the appearance will be like the plain type above. For movies with more interesting clip regions you will get a window such as the one below:

Additional options are available for this window type. If you specify **cmdKeyDraggable** in the options part of the OpenMovie command, then you can drag the window about by clicking and dragging inside the window while holding down the command key. You can always drag the window by clicking anywhere on the border and dragging. This can be a bit tough on a 1 pixel border. However, you can also specify a border width in the OpenMovie command. The **borderWidth** option allows you to specify a width between 0 and 6. Zero will give you a window with no border. In fact, the Borderless window type option uses the movieWDEF with a border width of zero.

```
QTMovie OpenMovie, MovieWDEF, "HD:Movies:AppleMovie", "10,10", cmdKeyDraggable,
borderWidth,2,noController
-- starts a movie in an window the shape of the movie AppleMovie, without a controller.  The border will be 2
pixels thick, and the window is cmd key draggable.
```

You can set the color of the MovieWDEF window border.

```
Set windowBorderColor of window "AppleMovie" to "45000,0,0" -- makes border red
```

The **windowBorderColor** property will only work for movieWDEF windows. The color is expressed as an RGB triplet (where each component is a value between 0 and 65535). If you want to have the window start out in a particular color, specify Invisible on OpenMovie, set the border color, and then do a show window.

The movieWDEF requires that a small stub WDEF (id#999) resource be in the stack. This resource is included in the QTMovie stack, but if the resource is not present the XCMD will create one on the fly and include it in the stack. The resource is six bytes long.

The QTEditMovie XCMD described below can be used to set a permanent clip region on a movie. To set a temporary clip on the movie you can set the **bitMapClip** property or send the **pasteBitMapClip** message. Both of these use a bit map to create a region that is used to set the movie's clip region. For example if you draw a solid black circle in the card window and then place a transparent button over it, you can execute the following lines to get a circular window movie:

```
QTMovie OpenMovie, MovieWDEF, "HD:Movies:NormalMovie", loc, cmdKeyDraggable, noController,invisible
if "Error" is in the result then <error handling>
```

```
set bitMapClip of window NormalMovie to the rect of btn circleButton
show window NormalMovie
```

Every pixel with in the specified rectangle will be used to create the region, so you need to be careful what rectangle you specify. The pasteBitMapClip message can be used to get the bitmap from the clipboard rather than having to have the bitmap visible on the card. One way to do the above without having the bitmap visible on the screen would be to have the button be opaque rather than transparent:

```
on ShowFunnyWindow
        set lockScreen to true
        choose select tool
        drag from topLeft of btn circleButton to botRight of btn circleButton
        domenu copy picture
        choose browse tool

        QTMovie OpenMovie, MovieWDEF, "HD:Movies:NormalMovie", loc,      cmdKeyDraggable,
noController,invisible
        if "Error" is in the result then <error handling>

        send PasteBitMapClip to window NormalMovie
        show window NormalMovie
end ShowFunnyWindow
```

The bitMapClip property and pasteBitMapClip message can also be used with Direct movies to create interesting shaped movies.

Track Oriented Properties

QuickTime movies are structured as a set of movie tracks, each track being of a particular type. The most common QuickTime movies contain one video track and one sound track. However, it is quite possible to have multiple video and sound tracks as well as text tracks, music tracks, and any other type that might be developed. The QTEditMovie XCMD described later has a number of facilities for manipulating individual tracks as well as a special track display mode.

Within QTMovie, there is a limited facility to get information about and temporarily manipulate tracks within a movie. You can find out how many tracks there are in the movie by getting the **numTracks** property. Many of the track properties or messages operate on the "current track". You set the current track by setting the **currTrackNum** property. The first track is track number one. You can find out the four character track type of the current track by getting the **currTrackType** property. Video tracks are 'vide', sound tracks are 'soun', text tracks are 'text', and music tracks are 'musi'. The following function returns the number of text tracks in a given movie:

```
function numTextTracks windowName
    put numTracks of window windowName into nTracks
    put 0 into textTracks
    repeat with i = 1 to nTracks
        set currTrackNum of window windowName to i
        if the currTrackType of window windowName = "text" then
            add 1 to textTracks
        end if
    end repeat
    return textTracks
end numTextTracks
```

Tracks in a QuickTime movie can be enabled or disabled. Disabled tracks are not played. You can selectively enable or disable tracks using the **enableTrack** and **disableTrack** properties. You set the value of the property to the number of the track you wish to enable or disable. Tracks can be combined into what are called alternate groups. When tracks are combined into an alternate group, at most one member of the group is enabled at a time. You can enable or disable a group of tracks by setting the **enableGroup** and **disableGroup** properties. You pass as a value to these properties the number of any track in the group. When you set disableGroup, all members of the group shared by the track number you pass in are disabled. What happens for enableGroup is somewhat less intuitive. The one appropriate track belonging to the group shared by the track number you pass in is enabled. For example, if you have a group of three sound tracks, one English, one French, and one Spanish and you set enableGroup with the number of the English track, the actual track enabled will depend on what the current movie language is. The current movie language is determined by QuickTime when the movie is opened by looking at the current system language. You can subsequently reset the language by setting the **movieLanguage** property to a particular region code. (Region codes are listed on page 14-133 of Inside Mac vol 6.) When you set the movieLanguage property, QuickTime will automatically enabled the appropriate track of any grouped tracks and disable all other tracks in the group unless all members of the group were already disabled in which case they all remain disabled. You can get a list of all the languages represented in a movie by getting the **movieLanguages** property. Another use of alternate groupings of tracks is to group video tracks according to certain quality and bit depth characteristics. For example, you can have one set of video tracks for color displays and an alternate set for black and white displays.

Every track is assigned a layer number that determines in what order it is displayed with respect to the other tracks. You can alter the layering of the current track by setting the **currTracklayer** property to a layer number. Layer numbers range from -32,768 to 32,767 with lower numbers being closer to the front. You can change the layer number of all members of an alternate group at once by setting the **currGroupLayer** property. All tracks that belong to the same group as the current track will have their layer changed. You can send the current track to the front or back by sending the **bringTrackToFront** or **sendTrackToBack** messages. As well, you can send all members current track's group forward or back by sending **bringGroupToFront** or **sendGroupToBack**.

You can get and set individual volume levels of sound tracks using the **currTrackAudioLevel** property. Levels range from 0 (silent) to 256 (full volume). With QuickTime 1.6 and Sound Manager 3.0, you can use higher values to amplify the sound.

Text Oriented Properties

Text tracks were introduced in QuickTime 1.5 (and substantially improved in QuickTime 1.6). The QTEditMovie XCMD allows you to add new text tracks to a movie. Included in QTMovie are properties and messages for extracting text, searching for text, and highlighting text. All of the text track functions will first look at the current track. If that is a text track then that will be the track used. If it is not a text track, then the first text track in the movie will be used.

To extract the text at a particular time in the movie you first set the **textSampleTime** property to the movie time you want. Then you get the **currTextSample** property. A string is returned representing the text at that time. Style information is not available. If you set textSampleTime to –1 to then currTextSample will use the movie's current time.

To search for text you first set the **findString** and **findFlags** properties. The findString property is simply the text you want to find. FindFlags is a value you get by adding together individual flag values depending on how you want the text to be searched. The flag values are:

>   1 - allow the current sample to be searched; otherwise start search at next sample
>   2 - make the search case sensitive
>   4 - search in the reverse direction
>   8 - wrap around search (when end hit, start at beginning of movie)
>   16 - use the offset into last found sample to begin the search (implies search current sample)

For example, if you wanted a case sensitive, wrap around search you would set the findFlags to 10 (2+8). Once you have set the findString and findFlags, you can send the **findNextText** message. The search is always started from the current movie time. The result will contain a string consisting of three numbers separated by spaces. The first number is the movie time of the found text; the second number is the duration of the text sample; and the third number is the offset into the text sample of the found text. For example suppose a movie contained the text sample "Fee fie foe fum" at time 600 with a duration of 100, and a current movie time of 0. The following script fragment will find the text and set the movie time to the found time (note that findNextText will not set the movie time for you):

```
set findString of window currWindow to "foe"
set findFlags of window currWindow to 0
send findNextText to window currWindow
put the result into info
if word 1 of info >= 0 then
     set currTime of window currWindow to word 1 of info
end if
```

In this case the result will be "600 100 9". If the text is not found, all three values will be -1.

By default the current track is the only track searched (or the first text track if the current track is not a text track). You can have findNextText search multiple tracks by setting the **searchType** property. Three values are allowed: 0 means search one track; 1 means search all enabled text tracks; and 2 means search all of the text tracks (enabled or not). When the search is complete, if the text was found, then the current track is set to the track where the nearest matching text was found.

You can highlight text by sending the **hiliteText** message. You first need to set up the highlighting by setting various highlight properties. The **textHiliteTime** property sets the movie time of the sample to be highlighted. The **textHiliteBegin** property sets the offset into the text sample of the beginning of the highlight. The **textHiliteEnd** property sets the ending offset of text to be highlighted. If you follow the previous script lines with the following lines the found text will be highlighted:

```
set textHiliteTime of window currWindow to word 1 of info
set textHiliteBegin of window currWindow to word 3 of info
set textHiliteEnd of window currWindow to word 3 of info + 3 -- number of chars in "foe"
send hiliteText to window currWindow
```

The highlight color will default to the system default. If you wish to choose your own highlight color you can set the **hiliteColor** property. As with other color properties, it is an RGB triplet. If you want to return to the default highlight color you need to set the **useHiliteColor** property to false. It is set to true automatically when you set the hiliteColor.

Movie Picts

You can copy an image of the current movie frame to the clipboard by sending the **copyFrame** message. Likewise you can copy the movie poster to the clipboard by sending the **copyPoster** message. If you prefer to write the image out to a pict file you can set the **copyFrameToFile** or **copyPosterToFile** properties. You supply the name of the file for the property's value. If you first set the **pictCreator** property then the pict file created by copyFrameToFile or copyPosterToFile will have its creator set to that value. The default value is "ppxi", the signature of the Picture Compressor application. For example the following script will create an Adobe Photoshop pict file of the movie at the given time:

```
on MakePhotoShopPict movieTime, fileName
    global currWindow
    set currTime of window currWindow to movieTime
    set pictCreator of window currWindow to "8BIM"
    set copyFrameToFile of window currWindow to fileName
end MakePhotoShopPict
```

Miscellaneous Properties and Options

You can set the **foreColor** and **backColor** properties to colorize the movie controller. This was more straight forward with the QuickTime 1.0 black and white controller. With the QuickTime 1.5 (and beyond) color controller, setting these properties may have weird effects. The values for these properties are RGB triplets.

Another OpenMovie initial option is **useCustomCLUT**. When this is specified, if the movie has a custom color look up table associated with it, that palette will be assigned to the window.

The **cacheMovie** property, when set to true, sets an internal QuickTime flag that causes movie data that has already been played to remain in memory longer. This may improve performance if you will be randomly accessing the movie, but memory may be used up more quickly.

The size of the movie file is available by getting the **fileSize** property. The size returned is in bytes.

You can obtain information about the current video track (or the first video track, if the current track is not a video track) by getting the **videoCompressorInfo** property. It returns a return delimited list of information about the video track. The first line is the codec type (eg: "rpza", "jpeg"). The second and third lines are the spatial and temporal quality settings (0-1023). The fourth line is the bit depth. The fifth line is the codec name (eg: Video, Photo).

For the really perverse, you can get the **movieHandle** and **movieController** properties. Returned are the actual handles to the movie and movie controller data. This is only useful if you plan to pass the value to a custom XCMD that expects a movie or movie controller handle.

The **version** property returns the date that the QTMovie XCMD was last compiled. You can also get this value by executing "QTMovie version" and getting the result.

Using the Movie Controller for Direct Movies

By default, there is no movie controller displayed when you start up a Direct movie. The reason for this

is that since the movie is not playing in an XWindow, mouse clicks are not sent to the movie. However, you can have a movie controller if you place a button behind the movie that sends the **mouseDown** message to QTMovie. You need to also pass a point in global coordinates with the mouseDown call. To get the movieController to show up, you need to call **showController**. For example, the following script opens a Direct movie with a controller. It assumes there is a rectangle style button on the card called movieButton, which it resizes to fit the movie and controller.

```
on OpenDirectMovie fileName
    global movieID
    QTMovie OpenMovie, Direct, fileName, topLeft of btn movieButton
    get the result
    if "error" is in it then <error handling>
    else put it into movieID

    QTMovie Direct, movieID, Get, movieRect
    put the result into r
    -- make the rect 1 bigger all around to make a frame for the movie
    -- Add 16 to the bottom to make room for the controller
    subtract 1 from item 1 of r
    subtract 1 from item 2 of r
    add 1 to item 3 of r
    add 16 to item 4 of r
    set rect of btn movieButton to r
    QTMovie Direct, movieID, showController
end OpenDirectMovie
```

The script for movieButton must convert the mouse location to global coordinates and then send it to QTMovie:

```
on mouseDown
    put the mouseLoc into pt
    add the left of card window to item 1 of pt
    add the top of card window to item 2 of pt

    global movieID
    QTMovie Direct, movieID, mouseDown, pt
end mouseDown
```

You can have a movie controller in Macromind Director by placing a similar mouseDown script in a cast member behind the movie:

```
on mouseDown -- ••• MacroMind Director script •••
    put the stageLeft + the mouseH into ptH
    put the stageTop + the mouseV into ptV
    put ptH & "," & ptV into pt

    global movieID
    QTMovie ("Direct", movieID, "mouseDown", pt)
end mouseDown
```

The DirectWindow Option for Direct Movies

An OpenMovie option specific to Direct movies is the **directWindow** option. If you specify directWindow followed by a window name then the Direct movie will appear in the named window rather than the card window. This could be useful if used with the Palette Maker feature of HyperCard

2.0.

The PlotPath Feature of Direct Movies

A fun feature of Direct movies is the ability to "paint" the movie onto the card window. You can do this by having a button behind a direct movie whose mouseDown script moves the button as the mouse is dragged and also resets the movieLoc of the movie to follow the button.

The **plotPath** message allows you to have the movie automatically painted across the window between two specified points. You need to first set up several properties before calling plotPath. The **pathStartPt** and **pathEndPt** properties specify the begin and end points in the card window along which to display the movie. The **pathStartTime** and **pathEndTime** properties specify the segment of the movie you want to play (default is the entire movie). The **pathNumFrames** property indicates how many steps there are between the start and end points. The **pathPlayFrames** property, if set to true, tells the XCMD to play the movie as the frames are splatted onto the screen, otherwise just the necessary frames are displayed. By default, a mouse click will cancel the plot. If you set the **abortPlotPathOnClick** property to false, then a click will not stop the plot.

If you set up these properties and then send the plotPath message, the specified frames of the movie will be splattered across your screen. For example you could have a button that contains the following script:

```
on mouseUp
        QTMovie OpenMovie, Direct, "MyHD:MyMovie", the rect of btn startBtn, Paused
        put the result into movieID
        if "Error" is in movieID then
                answer movieID
                exit mouseUp
        end if
        QTMovie Direct,movieID,Set,pathStartPt,the topLeft of btn startBtn
        QTMovie Direct,movieID,Set,pathEndPt,the topLeft of btn endBtn
        QTMovie Direct,movieID,Set,pathNumFrames,20

        QTMovie Direct,movieID,PlotPath
        QTMovie Direct,movieID,Dispose
end mouseUp
```

This would spread out 20 frames (evenly distributed throughout the movie) along a path defined by the top left corner of two buttons, startBtn and endBtn. Note that the final location will not necessarily exactly coincide with the end point, due to integral placements of the frames. If you set pathPlayFrames to true, then the intervening frames will be played at normal speed at the splattered positions. If audio is turned on, you will hear the movie.

The QTEditMovie XCMD

The QTEditMovie XCMD provides a variety of editing functions for QuickTime movies. The QTEditMovie window displays the movie along with a graphical representation of the tracks within the movie. This display allows you to select individual tracks which you can then cut, copy, and paste, as

well as perform many other functions.You also have control over collecting tracks into alternate groups. QTEditMovie allows you to add new sound tracks by capturing sound from an audio digitizer or copying from a sound resource. You can also add new text tracks and add text samples to the track.

The QTEditMovie Window

You open a new QTEditMovie window by executing the following command:

>   QTEditMovie fileName, windowType, location, [,options]

As with QTMovie, you specify the full path name of the movie file. You also supply a window type. The window type can be one of the following: Document, Windoid, TallWindoid, Plain, Dialog, or AltDialog. You can also supply the id of your own WDEF. The Borderless and MovieWDEF options of QTMovie are not supported by QTEditMovie. The location parameter must be a point expressed in the coordinates of the card window. There are two optional parameters: **newMovie** and **invisible**. If you specify newMovie, then a new, empty movie file is opened using the file name supplied. If there is a current file by that name it will be deleted, so be careful. If invisible is specified, then the window will initially be hidden. You can make the window visible by sending a show window windowName command or by setting the **visible** property of the window to true.
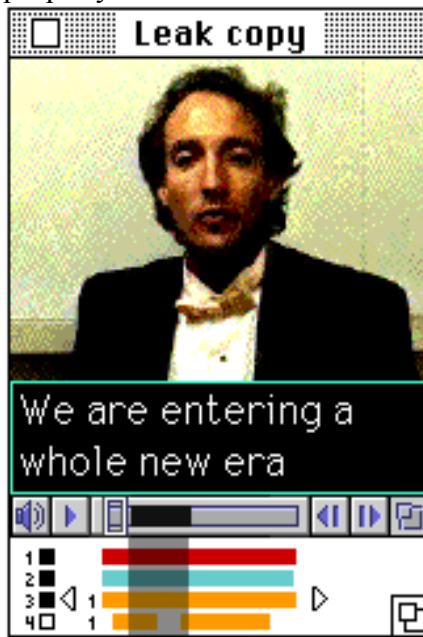
When you open the QTEditMovie window, there a few properties you can set that will affect the display. If you set the **displayTracks** property to true, then a graphical representation of the tracks within the movie will be displayed at the bottom of the window. The **displayTrackNums** property determines whether the tracks have their track number displayed. The **displayGroupNums** property tells whether to to display group numbers. The display shown above has all of these properties set to true. The grow icon in the lower right of the window allows you to grow or shrink the track display portion of the window vertically. If there are more tracks than can fit in the display, a scroll bar will appear, allowing you to scroll down to see all of the tracks. The grow icon in the movie controller is made visible when you set the **growable** property to true. Dragging this icon grows the movie rectangle and the window will resize itself appropriately.

The track display consists of colored representations of each track in the movie. Currently, video tracks are red, sound tracks are light blue, text tracks are orange, music tracks are green, and any other type is gray. The right extent of the track in the display indicates the duration of the track. Gaps in the track display show where no samples for that track exist. When a selection is made in the movie controller, the corresponding area in the track display is shaded. The small numbers on the very left of the display are the track numbers. The boxes to the left of each track are used to enable or disable each track. When the box is filled in, the track is enabled. You can click on an enable box to toggle its state. Note that if you enable or disable a displayable track (such as video or text) the movie rectangle may change. The window is immediately resized to reflect the new movie rectangle. This has the unfortunate consequence that the place you just clicked (the track's enable box) might no longer be under the mouse location. The small numbers immediately to the left of a track are the group number. Group numbers are only displayed for tracks that belong to an alternate group. The number assigned to a particular group is arbitrary and is used simply to be able to identify those tracks belonging to the same group. In fact, if there were three groups of tracks numbered 1 to 3, and the second group was "ungrouped", then the third group would suddenly become group number 2. You can make a track the *current track* by clicking on the track either in the movie itself (if it is displayable), or in the track display area. The current track is identified by the small stepper triangles to either side of the track. If the current track is a displayable track then it is outlined in the movie display. You can click on the stepper triangles to slide the track forward or backward in movie time. The details of this are discussed later. The current track is the target of many of the commands you can send that are discussed below. You can also set the current track by setting the **currTrackNum** property to the number of the desired track:

        set currTrackNum of window currWindow to 3

When you click in the movie to select a displayable track, it is outlined in blue. If you click and drag, you can change the location of the track within the movie rectangle. A gray outline of the track is displayed as you move the mouse. When you release the mouse button, the track will be placed at the new location. If you drag outside of the window, the window will be grown to fit the new dimensions of the movie.

Movie Control and Information in QTEditMovie

As with QTMovie, you can control the movie using the standard movie controller. The controller is always visible. The keyboard controls are also always on, which allow you to control the movie using the keyboard when the movie window is the active window. In addition, with QTEditMovie, the editing capabilities of the movie controller are activated, allowing you to make selections in the controller using the shift key. There are a limited number of properties and messages to control movie playback or get information about the movie.You can set the following properties:

        currTime- get or set (in the movie's time scale) the current time (does **not** pause)
        loop   - turns on or off loop mode (set to true or false)
        currSelection   - set the movie selection. Format: beginTime && endTime
        segmentPlayMode   - when set to true, only the current selection is played
        duration - returns the duration of the movie
        movieScale   - returns the movie's time scale

The following messages also control the movie:

        play   - play the movie

         pause   - pause the movie

Cut, Copy, Paste and More

There are a number of messages you can send to perform various editing functions. Some operate on the movie as a whole, while others only affect the current track. For a particular track the editing command may affect the whole track or only the current selection within the track. The **copy** message copies the current movie selection (all tracks) onto the clipboard. The same function is executed if you select Copy from HyperCard's Edit menu, if the movie window is the active window. Likewise, you can send the **cut** message. This will copy the current selection to the clipboard and clear it from the movie. The **clear** message will clear the selection. You can cut just the current visible frame by sending the **cutCurrFrame** message. The duration of the frame is automatically calculated for you. This affects all tracks at that time. The **copyTrackSelection** message will copy the selected portion of the current track to the clipboard. **CutTrackSelection** will cut the selected portion of the current track. The **copyTrack** message will copy the entire current track to the clipboard. The **cutTrack** message will copy the current track to the clipboard and then delete the track from the movie. When a track is deleted, its representation in the track display is removed, and the window size is adjusted, if necessary.

What all of these cut and copy messages actually place on the clipboard is a movie representing the copied portion of the source movie. For example if copyTrack is sent, then a one track movie is created and placed on the clipboard. The data that is used to display the track is not actually copied, only a reference to the data. The paste messages that follow only paste a reference to the original data. Thus, if you copy a track from one file and paste it into another, the second file will contain a reference back to the first file.

The **paste** message pastes the movie on the clipboard into the destination movie. (The destination movie is the movie belonging to the window to which the paste message is sent.) The pasted movie is inserted at the current time of the destination movie. If you do not want the clipboard contents inserted, but rather added in parallel, you can send the **add** message. New tracks are added to the movie to accommodate the clipboard movie. The **addScaled** message will also add in parallel, but the new tracks will be scaled to fit the current movie selection.

The following script will create a new file that is the merge of two given movie files:

```
on mergeMovies volume, file1, file2, newFile
    QTEditMovie volume & file1, windoid, "0,0", invisible
    QTEditMovie volume & file2, windoid, "0,0", invisible
    QTEditMovie volume &  newFile, windoid, "30,30", newMovie
    set currSelection of window file1 to 0 && duration of window file1
    send copy to window file1
    send paste to window newFile
    set currSelection of window file2 to 0 && duration of window file2
    send copy to window file2
    set currSelection of window newFile to 0 && 0
    set currTime of window newFile to duration of window newFile
    send paste to window newFile
    close window file1
    close window file2
    send saveChanges to window newFile -- see below
end mergeMovies
```

Note that in the example above, the two source movies' windows were made invisible so that only the merged movie is seen. The merged movie file created will be quite small. It will contain references to the two source movies.

Saving Changes

When you make a change using on of the editing commands, the change is not saved until you send the **saveChanges** message. This will update the movie resource of the file on disk. You can find out if a movie has changed by getting the **movieChanged** property. Alternatively, you can set the **autoSave** property to true, which will cause changes to the movie to be saved automatically when the movie window is closed. It nevertheless is a good idea to send saveChanges from time to time, just as you would when editing any document.

QTEditMovie has the same window close call feature back as QTMovie. If you set the **windowCloseMsg** property to the name of a handler in your card, background, or stack script, it will be called when the window is closed. For example, you can set up a handler that checks to see if any changes have been made before the window is closed:

```
on windowBeingClosed windowName
    if movieChanged of window windowName then
        answer "Do you want to save the changes to the movie" && windowName &"?"¬
        with "Don't Save" or "Save"
        if it = "Save" then send saveChanges to window windowName
    end if
end windowBeingClosed
```

Fun with Tracks

As stated previously you can set the current track either by clicking on it in the track display or movie rect or by setting the **currTrackNum** property. You can find out the type (eg: "soun", "vide", "text") of the current track by getting the **currTrackType** property. The number of tracks in the movie can be obtained by getting the **numTracks** property.

The stepper buttons that appear on either side of the current track can be used to slide the track forward or backward in time. The amount of time that a track shifts for each click of a stepper button is determined by setting the **trackShiftTicks** property. The value you set this to indicates how many ticks (sixtieths of a second) the track will be shifted in the appropriate direction. The default value is six ticks (one tenth of a second). If you hold down the mouse button over the stepper button, the track will continue to slide in the specified increment. For very fine control you can set the **trackShiftTime** property. For this you pass a time in the scale of the movie. For example if the movieScale property returned 600, you could set the trackShiftTime property to 2 to have each click on the stepper shift the track by 1/300 of a second. You can also slide the current track by setting the **slideTrack** property to the number of ticks you want the track shifted or by setting the **slideTrackTime** property to the amount in movie time by which you want to shift the track.

Alternate track groups are used to collect tracks together for which only one should appear at a time. Typically tracks are placed into alternate language groups or alternate quality groups. To place tracks into an alternate track group, you first set the **groupType** property to the type of tracks you are going to

group (eg: "soun", "vide", "text"). Then for every track in the movie of that type, you need to enable only those that you wish to belong to the alternate group. All other tracks of the same type must be disabled. You can enable or disable tracks by clicking on the enable box to the left of each track in the track display. You can also set the **enableTrack** or **disableTrack** properties, passing in the number of the track to enable or disable. To group the enabled tracks of the type previously specified send the **groupEnabledTypedTracks** message. If you need to group tracks of different types, you must enable only those tracks you want to group together and then send the **groupAllEnabledTracks** message. To ungroup an existing group of tracks, first set any member of the group to be the current track, then send the **ungroupTracks** message. All members of the group to which the current track belongs will be ungrouped.

Normally QuickTime attempts to enforce the alternate track rule of only having one track enabled in a group at a time. In QTEditMovie this automatic enforcement is turned off, thus allowing you to enable multiple tracks within a group while you are editing. However, QuickTime will still enforce the rule at certain times. When you set the movieLanguage property, any group of tracks with multiple languages will be subject to the alternate selection process. Also, if the bit depth of the display the window is playing on changes, the appropriate track within each group is chosen. You can force QuickTime to go through the alternate selection process by sending the **selectMovieAlternates** message.

To set the language of the current track set the **currTrackLanguage** property. You can also get this property to find out the track's current setting. The value for this property is a region code. A list of region codes appears on page 14-133 and 14-134 of Inside Mac, vol 6. You can set the movie's language by setting the **movieLanguage** property. This property is not saved with the movie. Movies always start out with the language of the System Software the Macintosh was booted on. You can set the quality of a track by setting the **currTrackQuality** property to a quality value. Bits 0-5 of the quality value correspond to bit depths 1-32. Bits 6 and 7 correspond to a quality level, 0 for draft, 1 for normal, 2 for better, and 3 for best. For example to set a track to support 16 and 32 bit pixel depths at quality level best the value would be 240 (3*64 + 32 + 16).

Every track is assigned a layer number that determines in what order it is displayed with respect to the other tracks. You can alter the layering of the current track by setting the **currTracklayer** property to a layer number. Layer numbers range from -32,768 to 32,767 with lower numbers being closer to the front. You can change the layer number of all members of an alternate group at once by setting the **currGroupLayer** property. All tracks that belong to the same group as the current track will have their layer changed. You can send the current track to the front or back by sending the **bringTrackToFront** or **sendTrackToBack** messages. As well, you can send all members current track's group forward or back by sending **bringGroupToFront** or **sendGroupToBack**.

Movie and Track Dimensions and Clipping

There are a number of properties that affect movie and track spatial characteristics. The **movieRect** property returns the bounds of the movie. This takes into account which tracks are enabled as well as any scale factor on the movie. You can find out the bounding rectangle of the current track by getting the **currTrackRect** property. You can also set this property to change a track's bounding rectangle.

If you set the **movieClipRect** property, the movie bounds will be clipped to the rectangle you pass in. The movie window will be resized to fit the new movie bounds. The **trackClipRect** property is used to set a clipping rectangle on the current track only. The **bitMapMovieClip** property is similar to the

bitMapClip property in QTMovie in which the rectangle passed in is used to locate a bit map painted on the card window from which the clipping region is calculated for the movie. The **bitMapTrackClip** property is used in the same way to set an arbitrary clipping region on the current track. If you pass zero instead of a rectangle for any of the clipping properties, the corresponding clip is cleared.

Miscellaneous Features

QTEditMovie allows you to set the poster and preview for a movie. The **posterTime** property sets the movie's poster. The **previewTime** and **previewDuration** properties set the movie's preview. All of these take a time in the movie's time scale as values.

You can change the name of the window with the **windowName** property. If you set the **dontDimController** property to true, then the movie controller is not dimmed when the movie window is inactive.

The **copyFramePict** message will copy the current movie image to the clipboard as a pict.

Capturing Live Audio

With QTEditMovie you can grab sound from an audio digitizer. A new sound track is created and added to your movie file. To add a new audio track to an existing movie you send the **grabAudioSoon** and **grabAudioNow** commands:

```
send releaseSound to window "Live Video" -- need to do this if QTRecordMovie XCMD was
                               previewing sound (see QTRecordMovie)
send GrabAudioSoon to window "My Movie"
send GrabAudioNow to window "My Movie"
```

The **soundStart** property defaults to zero, in which case the sound grabbing starts immediately after you send grabAudioNow. However if you set a value for it, the grabber will wait until that time to start grabbing. The time is expressed in system ticks (thus use "the ticks" from HyperTalk to determine a value). You can set **soundDuration** to how long you want the resulting audio track to be (in ticks) or if you specify the keyword *movieLength* rather than a time, then the sound duration will be exactly that of the current movie duration. If you set the **soundEnd** property then sound will be grabbed until that time is reached. You can use this to have the XCMD grab more sound than is specified by soundDuration, giving you a bit of "slop" for adjusting synchronization later. If you don't specify soundEnd, then soundDuration will be used to determine when to stop grabbing. SoundDuration will still be used to set the actual length of the new audio track.

The grabAudioNow message only gets the ball rolling for grabbing audio, the sound is grabbed during the window's idle time (which HyperCard automatically gives the window). Thus, returning from grabAudioNow does not mean the entire sound has been grabbed. You can be informed when the grab is complete by setting the **grabDoneMsg** property to the name of a handler to be called when the grab is done. This is similar to the call back properties described for QTMovie.

If you set the **playMovieWhileGrabbing** property to true before grabbing, then the movie will play as sound is recorded. You can lip synch your favorite video using this method. You should turn the movie's sound down before you do this so that the sound of existing tracks does not interfere with your

recording. You can have the audio grabbing stop on a mouse click by setting the **stopGrabbingOnClick** property before grabbing.

If you are recording from a microphone, you should set the **soundPlayThru** property to false before recording or you may get feedback through the Mac speaker.

If supported by your digitizer, you can set the **stereo** property to true to record in stereo. You can also set the **soundRate** property to either "11K", "22K", "44K", or 0. If you set it to zero the digitizer's default rate is used. If you set it to one of the named rates, the digitizer must support that rate.

Importing Data into a Movie

You can add a sound resource to a movie by first setting the **soundName** property and then sending the **addSoundResource** command:

        set soundName of window "My Movie" to ribbit
        send AddSoundResource to window "My Movie"

The sound resource needs to be in the current resource path (such as in the current stack).

You can also import sound or other data from a file using the **importFile** property. You set the property to the name of a file you wish to import. If the type of the file is compatible with one of the existing movie import components, then the data will be added to the movie. Currently there is support for importing sound files, AIFF files, PICS files, and text files. By default the data is added in parallel. If you want the data to be inserted into the file, you need to set the **importInParallel** property to false.

Flattening a Movie

As stated earlier, when you copy and paste from one movie to another, a reference to the original data is pasted into the file rather than the actual data. You can create a file that is self contained by *flattening* the movie file. If you send the **flattenMovie** message, a new movie file is created in which all of the data referred to by the window movie is copied into the new file. Data from deleted tracks is not copied, so you can reduce the size of a file if you have removed tracks from a movie. You can specify a destination file for the flattened movie by first setting the **destMovie** property to the path name of the file. If you do not specify a destName the name will the movie name with " flattened" appended.

By default, the movie's tracks' data are interleaved while the movie is being flattened. If you do not want this behavior, set the **dontInterleave** property to true. You can set the **activeTracksOnly** property to true to copy only currently enabled tracks. If you set the **addToDataFork** property to true, then flattenMovie will add the movie resource to the data fork, making the file compatible for non-Macintosh platforms.

Adding Text Tracks to a Movie

QTEditMovie has several properties and messages to add new text tracks, add text samples to the track, and specify highlighting for text. To add a next text track, first set the **textTrackRect** property to a rectangle within the movie for the new track. Then send the **addTextTrack** message. A new track will be

added. Since there are no text samples in the track yet, the track display for the new track appears blank. The new track is set to be the current track. Another way to add a new track is to set the **drawTextTrackMode** property to true. When this is set, a cross hair cursor appears as you move the mouse over the window. You can draw the bounds of the new text track directly into the window. After you draw the mode is set back to false.

There are a few ways to add text samples. Each of the add text messages uses the current movie selection as the start time and duration for the new sample. The text is added to the current track (or the first text track if the current track is not a text track). If you set the **text** property to a string of text and then send the **addText** message, that text will be added to the movie. Another way to add text is to send the **addSelectedText** message. In this case the text is obtained form whatever field text is currently selected within Hypercard. The default style for addText and addSelectedText is 12 point application font plain text. Before sending addText you can set up the style for the text by setting the **fontName**, **fontSize**, and **fontFace** properties. The fontFace property is the sum of a set of individual style values: bold is 1, italic is 2, underline is 4, outline is 8, shadow is 16, condense is 32, and extend is 64. You can also set the **justification** property to 0 for left, 1 for center, and -1 for right justification. The **textForeColor** and **textBackColor** properties can be set to choose the color of the text and the background. Their values are RGB triplets. These properties affect the entire text sample you are adding. To add multi-styled text you need to send the **addFieldText** message. You first set this up by setting the **textFieldName** property, identifying the HyperCard field from which the text will be extracted. The XCMD will first look for a card field by that name, then a background field. HyperCard fields can have multi-styled text. The text font, size, and style of the text within the field will be used in the text sample that is added to the movie.

You can specify that particular text within the sample being added be highlighted. If you set the **textHiliteBegin** and **textHiliteEnd** properties to the desired begin and end offsets into the text, then, when you send one of the add text messages, the indicated text will be tagged for highlighting. You can subsequently highlight additional text in the previously added sample by setting new values for textHiliteBegin and textHiliteEnd and the sending the **addHilite** message. You first need to set up a new selection in the movie controller to indicate the time and duration of the new highlighting. The effect of calling addHilite with new offsets set is to extend the existing text sample with the new text highlighted. For example, assuming you are highlighting text at is being spoken (in a sound track), your first add text message would add the entire text you want to display but the duration of the sample would be that of the first highlighted text. Then you would send the appropriate number of addHilite messages, each time extending the sample by the duration of the new text being highlighted. By default the system highlight color will be used when the text is displayed. You can specify a different highlight color by setting the **hiliteColor** property before sending an add text or addHilite message. To go back to the default color, you need to set **useHiliteColor** to false. It is set to true when you set the hiliteColor property.

The text media handler defines a set of flags that can be set for a text sample. You can set the **textFlags** property before calling one of the add text messages. The flag values are:

> DontDisplay = 1 Don't display the text
> DontAutoScale = 2   Don't scale text as track bounds grows or shrinks
> ClipToTextBox = 4   Clip update to the textbox
> ShrinkTextBoxToFit = 16 Compute minimum box to fit the sample
> ScrollIn = 32   Scroll text in until last of text is in view
> ScrollOut = 64   Scroll text out until last of text is gone (if both set, scroll in then out)
> HorizScroll = 128   Scroll text horizontally (otherwise it's vertical)

ReverseScroll = 256   vert: scroll down rather than up; horiz: scroll backwards (justification dependent)
ContinuousScroll = 512   new samples cause previous samples to scroll out
FlowHoriz = 1024   horiz scroll text flows in textbox rather than extend to right
DropShadow = 4096   display text with a drop shadow
AntiAlias = 8192 attempt to display text anti aliased
KeyedText = 16384   key the text over background

Add together the value you want set and set textFlags to the result. The scrolling properties have an optional scroll delay feature. You can set the **scrollDelay** property to a time value indicating the delay. This will have no effect unless you set the scrollIn or scrollOut (or both) flag. These flags are described in more detail in the text media section of Inside Macintosh: QuickTime.

By default the text that is added is flowed in a text box that is inset by 2 from the bounds of the text track. You can set your own text box by setting the **textBox** property. The rectangle you pass is relative to the top left of the track itself. For example, if you wanted the text box to fit the track bounds exactly, you would set the text box to the same size rectangle you used when you set textTrackRect before calling addTextTrack, but offset to 0,0. To go back to the default text box, you need to set **useTextBox** to false. It is set to true when you set the textBox property.

Undo

QTEditMovie supports undo for most of the editing operations described thus far. To undo the previous operation, simple send the **undo** message. If you send undo again, the operation will be redone.

The QTRecordMovie XCMD

The QTRecordMovie XCMD is used to connect to video digitizers to display live video in a window. You can then capture the video to make QuickTime movies. There are two ways to capture movies: controlled grab and live grab. Both methods are described below. You can also grab and compress a still picture and save it into a picture file.

To open a live video window, you send the QTRecordMovie command as follows:

QTRecordMovie windowName, windowType, windowRect, growable, connectToAudio, videoStandard, videoInput [,options]

The XCMD will search for a video digitizer board and, if it finds one, will display video in an XWindow specified by windowName, windowType, and windowRect. The windowType parameter follows the same conventions as for the QTMovie XCMD. WindowRect is given in card local coordinates. A typical window title may be "Live Video"; the title for the movie to be recorded is specified by sending a message to the window. If you specify true for growable, then you can resize the window by clicking in the lower right corner and dragging. By default the aspect ratio of the window will be maintained. If you want to arbitrarily resize the window hold down the shift key while you drag the the lower right corner. If you specify true for the connectToAudio parameter, then the XCMD will attempt to connect to an audio digitizer. The default is to not connect to an audio digitizer. If you do specify true, the sound will play through the Mac speaker while previewing video and while doing a live video and audio grab. You can suppress audio play through during preview or live recording by setting the **soundPlayThruPreview** and **soundPlayThruRecord** properties to false. You may wish to do this for

live grab from a microphone to avoid audio feedback. The videoStandard is an optional parameter that is provided so you can force the digitizer to use one of the specified standards: "ntsc", "pal", or "secam". The standard chosen must, of course, be supported by the digitizer. With the optional videoInput parameter you can specify which input of the video digitizer to use. You can get a list of the inputs by executing:

```
QTRecordMovie videoInputList
get the result
```

The result will contain a list of inputs for the video digitizer. Each element in the list will be of the form "inputFormat inputNumber", for example COMPOSITE 1, SVIDEO 2, COMPONENT 1. You can pass one of these strings as the videoInput parameter when you open the video window. If you do not specify videoStandard or videoInput, the digitizer's default is used. Once the window has been opened, you can change the video input by setting the **videoInput** property. If you want to get the video input list after the video window is opened, you need to get the **videoInputList** property, which will return the same list as above.

Likewise, you can get a list of available digitizer cards by either executing QTMovie VideoCardList or by getting the **videoCardList** property after the video window is opened. You can then set which card to use by setting the **videoCard** property.

The only optional parameter is **invisible**. If this is specified then the window will initially be hidden. This can be useful in conjunction with the cropWindow message discussed later. If you wish to have the window initially invisible, but do not want to specify either the videoStandard or videoInput parameters, you can pass *empty* for those parameters and the default standard and/or input will be used. You can show the window either by send show window or by setting the **visible** property to true. You can set the **windowCloseMsg** to the name of a handler in your stack to be called when the window is closed.

Video Properties

[The following paragraphs describe how you can change many of the video and audio properties from a script. Some new additions to the XCMD makes many of these properties unnecessary. The **ShowSoundDialog** and **ShowVideoDialog** commands bring up standard QuickTime dialogs for selecting the various video and audio options that a particular digitizer provides. You can save the settings you choose using the **SaveSoundPrefs** and **SaveVideoPrefs** properties the property value you specify is the name under which it will saved (as Psnd and Pvid resources).
You can later restore the settings by using the **RestoreSoundPrefs** and **RestoreVideoPrefs** properties, specifying the same name you used when you saved the settings.]

There are a set of seven properties that affect the live video display. They are **hue**, **saturation**, **brightness**, **sharpness**, **contrast**, **black Level**, and **white Level**. The range for these properties is 0 to 65535. If you try get the current value of the property and -1 is returned, then that property is not supported by the digitizer.

Often the default video rectangle used by the digitizer contains some garbage at one or more borders of the image. You can adjust the video rectangle to account for this. First, you can get the **maxRect** property to find out the maximum rectangle that you can set the video rectangle to. You can also get the **videoRect** property to find out its current setting. You can then set the **videoRect** to some rectangle within the maximum rect. If you send the **resetVideoRect** message, the video rectangle will be reset to its default. MaxRect and videoRect are independent of the current window size. Typically MaxRect may be something like "0,0,640,480" and videoRect is some rectangle contained within. The actual size of the image is determined by the size of the video window. Thus, when you adjust videoRect you are indicating what portion of the maxRect to display. You are not changing the size of the image.

The following script will shift the whole image up or down one pixel (note that the "pixel" in question is in the coordinate space of the maximum rectangle, thus if you are viewing a quarter screen image, you may need to shift the videoRect by two pixels to see a one pixel adjustment in the image you are viewing):

```
on ShiftImage direction
        if direction = "up" then put -1 into bump
        else if direction = "down" then put 1 into bump
        else exit ShiftImage

        put videoRect of window "Live Video" into vRect

        add bump to item 2 of vRect
        add bump to item 4 of vRect

        set videoRect of window "Live Video" to vRect
end ShiftImage
```

If you set the growable option to true when you opened the video window, you can click and drag in the lower right portion of the window to resize it. You can also change the window size by setting the **windowSize** property. You pass a point indicating the new width and height.The **loc** property is used to change the location of the window. This is a point in global coordinates. You can also drag the window by its title bar, or if there is no title bar, you can hold down the control key while clicking in the window to drag it to a new location.

You can freeze the video by sending the **videoOff** message and turn it back on by sending the **videoOn** message. You can also freeze and unfreeze the video by option/clicking in the window. You can find out the current freeze state by getting the **videoOn** property. True means it is on. Normally, when HyperCard is made inactive (another app is selected), the live video is automatically frozen. If you set the **keepVideoOn** property to true then the video will stay on when HyperCard is inactive.

If you hold down the command key and drag in the video window, you can select a portion of the video window to be grabbed. As you drag, a rectangle will appear in the window. The current coordinates of the rectangle are displayed in the upper left of the window. When you let go of the button, the window will show video only in the rectangle you selected, surrounded by a gray region. You can then move the cropping rectangle about by clicking in the center and dragging it about. You can resize the selection by clicking in any of the corners and dragging that corner. If you resize the video window (by dragging the lower right corner), the cropping rectangle will be proportionally resized. If you command/click without dragging, the cropping rectangle will be cleared. You can also set the cropping rectangle by setting the **cropRect** property of the window.
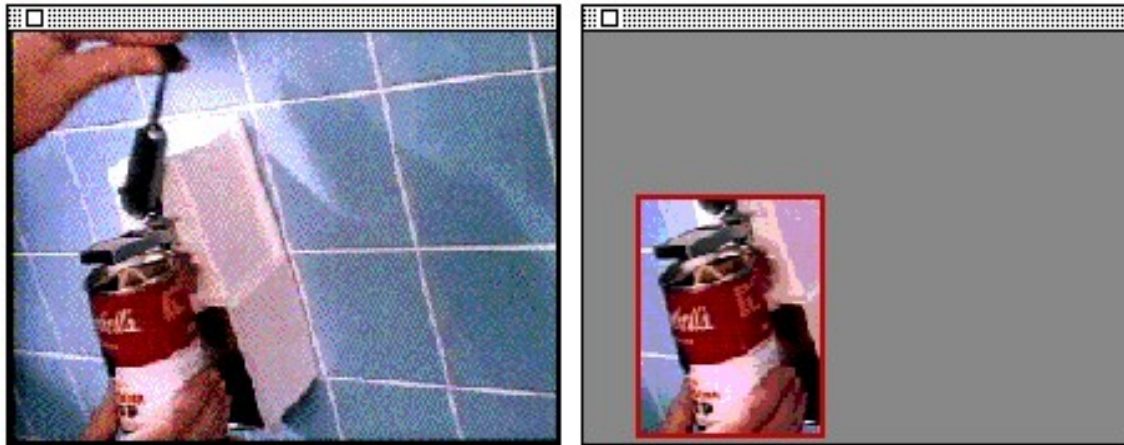
If you want the video window to just fit the cropping rectangle you can set the **cropWindow** property to true. The window will be resized and relocated to fit the crop rectangle.
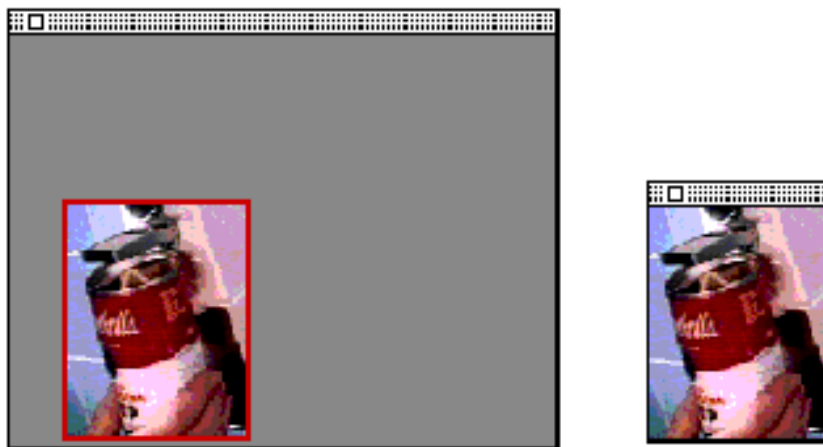
The following script will open a hidden video window, set the crop rectangle, and then set cropWindow before showing the window. It uses the rects of two transparent buttons on the card, wRect which is the full size of the video and cropBtn, which is a smaller button within the bounds of wRect.

```
on OpenCroppedWindow direction
        QTRecordMovie "Live Video", plain, rect of btn wRect, true, true, empty, empty,invisible
        if the result is not empty then <error handling>

        set cropRect of window "Live Video" to rect of btn cropBtn
        set cropWindow of window "Live Video" to true
        show window "Live Video"
end OpenCroppedWindow
```

While the cropWindow property is true, you cannot change the cropping rectangle.

Due to limitations in some digitizers and for compression optimization, the location and size of both the video window and the cropping rectangle have certain constraints. The location of a displayed video rectangle must start on an even scan line of the digitizer's monitor. Note that in global coordinates this may be odd or even depending on how the digitizer's monitor is offset from the main screen. When resizing the video window the size is gridded to be a multiple of four in the horizontal and vertical

dimensions, unless you hold down the shift key. Keeping the dimensions to multiple of four allows compression and decompression to work much faster in many cases. These constraints are enforced by the XCMD, so you do not need to be concerned about it other than to understand why the window location or size may not show up exactly as you specified.

Capture Properties

Before starting a video capture, you need to set a variety of properties affecting the capture. The **fileName** property specifies the new movie file to create. By default, if a file by that name already exists, it is first deleted. You can set the **deleteFile** property to false to indicate that it should not delete the file, in which case an attempted capture will fail. The creator for the new file is set to "TVOD" which is the creator type for the MoviePlayer application. You can specify a different creator by setting the **movieCreator** property to whatever four character type you want.

To set the compressor to use for the capture, you can first find out what compressors are available by getting the **codecList** property. This returns a list of available compressors by name. You can get a list of compressors by type by getting the **codecTypes** property. For example, the Video codec's name is "Video" and type is "rpza". You can then set the compressor to use by its position in either list by setting the **codecNumber** property. If the Video compressor was 7th in the list you would set codecNumber to 7 to choose that compressor. You can also set the compressor by setting the **codecType** property (eg: "rpza", "jpeg", etc). If there is more than one compressor of that type, QuickTime will choose one for you. Another way of getting the codecList is to execute:

```
QTRecordMovie codecList
get the result
```

This has the advantage that you do not need to have the video window open to get the list. You can get a list of bit depths that are supported by a particular compressor in a similar manner:

```
QTRecordMovie depthList,codecNumber
get the result
```

As before, the codecNumber is the codec's position in the codecList. The possible depth values are 1,2,4,8,16,24,32,33,34,36, and 40. Note that 33,34,36,and 40 are actually 1,2,4,and 8 bit gray level. You can set the depth at which to capture by setting the **movieDepth** property.

The **pictureQuality** setting controls the spatial quality of the capture. The **motionQuality** setting affects the temporal quality. Both of these have a range of 0-1023. If you want to design your own quality input interface, the QuickTime values for certain fixed quality levels are available as read only properties: **minQuality**, **maxQuality**, **lowQuality**, **normalQuality**, and **highQuality**. MotionQuality only takes effect if you have set the **frameDifferenced** property to true. If you specify that a movie be frame differenced, you can set the **keyFrameRate** property. This will indicate the minimum gap between key frames. Additional key frames may be added by the compressor if it deems it necessary.

The **frameRate** property controls what the movie's frame rate will be for controlled grabs. Typical values are in the 10-15 frames per second range. The rate for live grabs are whatever the live grab was able to achieve. If you set **throttleLiveGrab** to true, then the XCMD will attempt to constrain the live grab rate to that of the **frameRate** property.

The dimensions of the movie are determined by the size of the window (or crop rectangle if the window is cropped). Some digitizers may not display reduced sized images as well as one might want. The **grabSize** property allows you to set a size at which the image will be grabbed (generally something like quarter screen) and have it be copied into the smaller size you want the movie to be. The video window will automatically resize during the grab process to the specified size. If you use this feature, you can turn it off by setting the grabSize to 0,0. The default is that it is turned off. The grab process is slowed down considerably when grabSize is used.

Audio

If you specified true for the connectToAudio option when opening the video window, then QTRecordMovie will attempt to connect to whatever sound digitizer is selected in the Sound control panel. The sound will play through the Macintosh speaker while viewing live video unless you set the **soundPlayThruPreview** property to false. You can have the sound turned off only during recording by setting **soundPlayThruRecord** to false. You can also set the **audioLevel** property (0-256) to raise or lower the volume of the incoming sound.

To disconnect from the audio digitizer, send the **releaseSound** message. This is necessary if some other XCMD (for example QTEditMovie) needs the audio digitizer while the live video window is open. You can reconnect by sending the **startSound** message. You can also send this to connect to an audio digitizer if you initially had set false for the connectToAudio initial option.

For a live grab, if supported by your digitizer, you can set the **stereo** property to true to record in stereo. You can also set the **soundRate** property to either "11K", "22K", "44K", or 0. If you set it to zero the digitizer's default rate is used. If you set it to one of the named rates, the digitizer must support that rate.

Live Grab

To grab frames as fast as possible you first send the **liveGrabPrep** message. The XCMD prepares for a live grab. A subsequent **doLiveGrab** message starts capturing frames immediately into the movie file. If you are connected to an audio digitizer, either by setting the connectToAudio initial option to true or by sending the startSound message, then sound will be captured along with the video. If you don't want to capture sound, send the releaseSound message before starting the live grab. You can set either the **maxGrabTime** or **maxGrabTicks** property to limit the duration of the live grab. MaxGrabTime is expressed in seconds, maxGrabTicks in ticks (sixtieths of a second). By default the grabbed frames are written out to the disk. To make the grab go faster, you can set **grabToRAM** to true. Frames will now be grabbed to RAM at a faster rate, but the capture will halt when RAM is filled. The file will still be properly written after the capture halts.

By default the live grab can be halted by clicking the mouse during the grab. If you set **stopGrabbingOnClick** to false, then clicking will not halt the grab.

Controlled Grab

A controlled grab differs from a live grab in that you decide when a frame is grabbed. Audio is not

grabbed for controlled grabs. To start a controlled grab you send the **startControlledGrab** message. You can then send **grabOneFrame** at any time. Each time you send grabOneFrame a new frame is added to the movie. When you are done, send **finishControlledGrab**. For example, if you had XCMD's that controlled a laser disk, you could record frames from it by executing the following script:

```
on GrabSomeFrames startFrame, endFrame
        vidSearch startFrame
        send StartControlledGrab to window "Live Video"
        put startFrame into currFrame
        repeat while currFrame <= endFrame
                send GrabOneFrame to window "Live Video"
                vidStep
                put vidFrame() into currFrame
        end repeat
        send FinishControlledGrab to window "Live Video"
end GrabSomeFrames
```

This would grab every frame of the source movie. You may want to grab every other frame to save space and be at a more reasonable movie rate. At whatever rate you decide to grab, you must tell the XCMD by setting the frameRate property. It is expressed as frames per second, hence you would set it to 15 if you were grabbing at that rate. Some video disks (usually CAV disks from a film source) present still frames at 24 frames per second, hence grabbing every other frame would amount to 12 frames per second and you would need to set the value accordingly.

If you set **beepOnGrab** to true, then you will hear a beep whenever grabOneFrame is finished (which could possibly take a long time depending on the size of the frame you are grabbing and the speed of the compressor you chose).You can also set the **grabCompleteMsg** to the name of a handler in your stack to be called when grabOneFrame is complete.

Depending on the digitizer, if you are capturing video in a script loop as in the example above, the video window may only be updated when you send grabOneFrame. To be able to see the video while in a loop, you may need to send **Idle** to the window inside of the loop.

Special Features for Stop Frame Movie Making

You can set up your stack to have a button that sends the grabOneFrame command when you press it. With this, you can make your own animations. If you send **showController** to the window while making a controlled grab movie, the movie grabbed thus far will play in the window with a standard movie controller. You can navigate about in the movie using the controller. When the movie hits the end, the window automatically goes back to live video, allowing you to preview the next frame before you actually grab it. You can continue to use the controller and the live video will go off automatically. You can continue to add frames to the movie and use the controller alternately. When you are using the controller, you can pause at any frame you have captured so far and send the **cutCurrFrame** command to cut any frames you didn't like. If you send finishControlledGrab, you can still add to the movie by setting the **doAppend** property to true before again calling startControlledGrab. If doAppend is false, startControlledGrab will replace the file.

If the **showPrevFrameWindows** property is set to true, then two windows, each a quarter size of the main window, will appear to the right of the main window. As frames are added to the movie, these windows will display the last two frames grabbed, so that one can use them as a guide while setting up

the next frame. These windows can be dragged about by clicking and dragging inside the window.

Grabbing Still Pictures

You can grab a still picture to a picture file by sending the **grabPict** message. The picture will be the size of the window (or crop rectangle if it is set). The picture will be compressed using the same compression properties you set for capturing movies. It will written out to the file specified in the fileName property. If you first set the **pictCreator** property then the picture file created by grabPict will have its creator set to that value. The default value is "ppxi", the signature of the Picture Compressor application.

The QTPict XCMD

The QTPict XCMD performs a variety of Picture related utilities including displaying a picture on a card, compressing pictures, and allowing control over the clipping region of the card window.

Displaying Still Pictures

        QTPict DisplayPict, name, location, source, [,options]

DisplayPict is used to display a still picture (compressed or uncompressed) directly onto the HyperCard screen. To display pictures into an XWindow, use the built-in Picture XCMD provided with HyperCard 2.0. Name is the name of the Pict file or resource you wish to display. Location is given in the coordinates of the card window. It can be either a point or a rect. If a rect is supplied, the picture will be scaled to fit. If your picture has a mask associated with it, the picture will display using the mask. (Creating a picture with a mask is discussed below.) Source is either *file* or *resource*. The three optional parameters are **thumbnail, clipTo,** and **forceOffscreen.**Thumbnail applies to picture and movie files. If thumbnail is chosen, then a thumbnail (aka preview) of the picture or movie file is displayed. If the file does not already have a thumbnail resource, then one will be created for it and installed in the file. If the clipTo option is chosen, then the next parameter *must* be a rectangle. The rectangle specifies an area to which the displayed picture will be clipped. If you specify forceOffscreen, then the picture will drawn to an offscreen buffer first. This might be desirable for slow drawing images.

DisplayPict, like Direct movies, is a volatile operation. It simply blasts the pict directly onto the card window with HyperCard being none the wiser. As a result any operation that requires HyperCard to refresh all or part of the card will cause the picture to be erased. Use of the clipping commands below can mitigate this somewhat by preventing HyperCard from drawing over your pict, but it will not save you from dialog boxes or other windows erasing the pict.

If you get the result after DisplayPict, it will contain the rectangle of the drawn picture. You can call **QTPict PictBounds** with the same parameters as DisplayPict to get the bounds without displaying the pict.

Getting Available Compressors

```
QTPict CodecNames
put the result into codecNameList
QTPict CodecTypes
put the result into codecTypeList
```

CodecNames returns a return delimited list of codec names, that can be used for building a menu. CodecTypes returns a comma delimited list of the corresponding four character codec types in the same order as the codec name list. You can then use the following button script to choose a codec for the CompressPict command.

```
on mouseDown
        global codecNameList, codecTypeList, chosenCodec

        put PopUpMenu(codecNameList, 0, bottom of me, left of me) into itemNum
        if itemNum > 0 then
                put item itemNum of codecTypeList into chosenCodec
        end if
end mouseDown
```

Compressing Still Pictures

```
QTPict CompressPict, name, source, quality, codec
```

CompressPict is used to compress a picture resource or file. Name is the name of the resource or file you wish to compress. Source is either *file* or *resource*. Quality is a value between 1 and 1023. The default is 512. Codec is a four character compression type, which can be obtained by the method shown above. The default is "rpza", the Video compressor. The resulting resource or file will be called *name.qn* where name is the name you passed in and n is the quality level. [For now, if a resource is found with the same name, it is replaced; if a file is found with the same name it is not replaced.]

Capturing Screen Bits to a Picture File

```
QTPict ScreenBitsToPictFile, global rect, fileName
```

ScreenBitsToPictFile will copy whatever bits are on the screen within the global rectangle specified and copy them into a new picture file specified by fileName.

Converting a Pict resource to a Pict file

```
QTPict PictRsrcToFile, name
```

Since I couldn't find an application that would allow me to save a picture file with a mask that DrawPicture understands, I put the PictRsrcToFile command in as a utility routine. If someone can show me an easier way to do this, I may take it out. Thus the ugly process of creating a picture file with a mask (i.e. a "cutout") is the following: Create the picture you want to cut out. With a painting program, such as PixelPaint or Studio 32, lasso the part you want as your cutout, then copy it and paste it into the

Scrapbook. [Not every application will preserve the mask when you do this; eg Adobe Photoshop.] Then go into ResEdit (I told you it was ugly!), open the Scrapbook and your stack, find the picture resource, and copy and paste it into your stack. You can now execute the PictRsrcToFile command on the resource and the picture file will be created.

Converting a Pict file to a Pict resource

        QTPict PictFileToRsrc, fileName [,resourceName]

This will take the named pict file and convert it into a resource in the stack. You can specify a name for the resource, or the file name (minus the path) will be used.

Getting the Screen Depth of the screen the Card Window is On

        QTPict GetScreenDepth
        get the result

The result will contain the pixel depth of the deepest screen that the card window spans.

Getting a File's Size

        QTPict FileSize,fileName
        get the result

The result will contain the size in bytes of the file specified. Useful in determining the amount of compression the CompressPict command accomplished.

A Few Convenient But Dangerous Clipping Commands

        QTPict ClipTo, <rect>
        QTPict DiffClip, <rect>
        QTPict UnionClip, <rect>

These are used to futz with the clipping region of the card window. You may wish to use these in conjunction with DisplayPict or Direct movies, since the image they put on the screen can be erased at the whim of HyperCard when the card gets updated. ClipTo specifies a rectangle to which you want the card clipped. DiffClip will remove the given rectangle from the clipping region. UnionClip will add the given rectangle to the clipping region. For example, you might display a pretty little color picture on your card with a background button behind it. If you use DiffClip to remove the area of the pict from the clipping region of the card, then the color pict will not disappear when you move from card to card. But be careful – if you decide to suddenly go elsewhere, such as to the home card, the clipping region is still in the odd state you set it to. You may wish as a safety device, to have the following script called from the closeStack script of any stack that plays with the clip region.

```
on AllClip
        QTPict ClipTo, "0,0,1280,1280"
end AllClip
```

Another convenient routine is:

```
on NoClip
          QTPict ClipTo, "0,0,0,0"
end NoClip
```

XCMD Version Information

To find out what version of each XCMD you have, you can call the XCMD with the first (and only) parameter being **version**. The value of the result will have the date when the XCMD was built. For QTMovie, QTRecordMovie, and QTEditMovie, you can also ask for the version property of the window. Examples:

```
QTPict version
put the result into whatVersion
convert whatVersion to seconds
if whatVersion < neededVersion then answer "get a newer version"
-- neededVersion is a saved value that has already been converted to seconds


answer the version of window "live video"
```

Appendix

This is a comprehensive list of properties and messages for the QuickTime XCMDs QTMovie, QTEditMovie, QTRecordMovie, and QTPict as of 4/26/94.

QTMovie
        Forms
                QTMovie OpenMovie,<window type> or Direct,<fileName>,<loc>[,options...]
                QTMovie Direct,<movieID>[,options...]

        OpenMovie Options
                Badge -- show badge
                BorderWidth,width -- for MovieWDEF window; set borderwidth (0-6)
                ClipTo,rect -- Clip to specified rectangle
                CloseOnFinish -- close movie window when finished
                CmdKeyDraggable -- for MovieWDEF window; allow drag with cmd key down
                DirectWindow,wName -- Alternate window for the Direct movie to appear in
                DocumentLayer -- Force window into document layer (defaults to palette layer)
                DontPaintWhite -- Don't erase window before displaying
                FastIdle -- Don't return from idle until OS event occurs
                Invisible -- Start with the movie window hidden
                Loop -- Start in Loop mode
                LoadIntoRAM -- Load movie into RAM before playing
                Mute -- Start Muted

NoController -- show with no controller
Palindrome -- Start in Palindrome mode
Paused -- start paused
SeeAllFrames -- Show all frames while playing (no audio)
ShowGrowBox -- show movie grow box in controller
ShowPoster -- Show movie poster
UseCustomCLUT -- use Movie's color table, if appropriate for the screen depth

Direct Options
Dispose -- Call when done with movie
Get,<propName> -- Call to get a property
Idle -- Must be called in Idle routine for movie to run
MouseDown,<global point> -- Call from mouseDown handler
in button behind controller to use controller in direct movies
PlotPath -- Plot Frames along path specified above
Set,<propName>,<value>-- Call to set a property

WindowTypes
AltDialog
Borderless
Dialog
Document
MovieWDEF -- Window shape determined by movie's clip region
Plain
TallWindoid
Windoid
<wdef id> -- use your own wdef id

Set Properties
ActiveMovie <movieID> -- play previously queued movie
AudioLevel <0-256> -- sets movie volume
Badge <true/false> -- show badge when no controller
BackColor <rgb triplet> -- Set background color of movie controller
BitMapClip <rect> -- Use bit map from card at rect to set clip of window
CacheMovie <true/false> -- set movie hint that keeps movie data in memory
ClipRect <rect> -- set movie's clipping rectangle
CloseOnFinish <true/false> -- close window when end of movie hit
CopyFrameToFile <file name> -- create pict file of current frame
CopyPosterToFile <file name> -- create pict file of poster frame
CopyPreviewToFile <file name> -- create pict file of preview frame
CurrGroupLayer <layerNum> -- set layer of all tracks in currTrack's group
CurrTime <time in movie's scale> -- positions movie at specified time
CurrTrackAudioLevel <0-256> -- set currTrack's volume
CurrTrackLayer <layerNum> -- set currTrack's layer
CurrTrackNum <trackNum> -- set "currTrack" to trackNum
CursorMsg <handler name> -- call when cursor over window
DeleteQueuedMovie <movieID> -- delete queued movie from list
DisableGroup <trackNum> -- Disable all tracks in trackNum's group

DisableTrack <trackNum> -- Disable track trackNum
DontInvalOnClose <true/false> -- Don't update card when window closed
DontPaintWhite <true/false> -- Don't erase window before displaying
EnableGroup <trackNum> -- Enable appropriate track in trackNum's group
EnableKeys <true/false> -- Allow keyboard movie control when window active
EnableTrack <trackNum> -- Enable track trackNum
FastIdle <true/false> -- turn on the fast idle option
FindFlags <flags> -- set flags for FindNextText
FindString <text string> -- set string to find for FindNextText command
ForeColor <rgb triplet> -- Set foreground color of movie controller
HiliteColor <rgb triplet> -- set hilite color for HilteText command
Loop <true/false> -- set loop mode
MouseDownMsg <handler name> -- call when mouse clicked in movie
MovieRect <rect> -- set movie's rectangle
MovieLoc <pt> -- set movie's loc within window/card
MovieControlMsg <handler name> -- call when certain movie controls change
MovieLanguage <region code> -- set movie's language
Mute <true/false> -- mute the movie
NewMovieFile <fileName> -- play new movie in window
Palindrome <true/false> -- set palindrome (back and forth) mode
PictCreator <OSType> -- file creator type for CopyToFile cmds Default: 'ppxi'
QueuedMovie <fileName> -- place movie file in queue (movieID in the result)
Rate <fixed num> -- set movie play rate (takes effect when movie is playing)
ReplaceTime <time> -- start point for new movie in NewMovieFile/ActiveMovie
StatusMsg <handler name> -- call when error detected while playing
SearchType -- 0: search one track; 1: search enabled tracks; 2: search all tracks
SeeAllFrames <true/false> -- ensure all frames shown (audio shut off)
SegmentEnd <time> -- set end time for segment play
SegmentPlay <true/false> -- go into segment play mode
SegmentStart <time> -- set start time for segment play
TextHiliteBegin <offset> -- set text offset start for HiliteText command
TextHiliteEnd <offset> -- set text offset end for HiliteText command
TextHiliteTime <time> -- set movie time for HiliteText command
TextSampleTime <time> -- set time for CurrTextSample (-1: use curr movie time)
TimedCallBack <handler name && time> -- call when time reached
UseHiliteColor <true/false> -- set back to false to use default hilite color
Visible <true/false> -- show hide the window
WindowBorderColor <rgb triplet> -- Set border color for MovieWDEF window
WindowCloseMsg <handler name> -- call when window closed
WindowLoc <local pt> -- set new position for window
WindowName <name> -- rename the movie window (default is movie file name)
WindowRect <local rect> -- set new position/size for movie window

Direct Only Properties
AbortPlotPathOnClick <true/false> -- Abort PlotPath when mouse is clicked
EraseOnMove <true/false> -- Erase old when new position set
PathStartPt <pt> -- starting point for PlotPath command
PathEndPt <pt> -- end point for PlotPath command

PathStartTime <time> -- movie start time for PlotPath command
PathEndTime <time> -- movie end time for PlotPath command
PathNumFrames <value> -- # of frames to display for PlotPath
PathPlayFrames <true/false> -- play movie during PlotPath

Messages

BringGroupToFront -- Bring all tracks in currTrack's group to front layer
BringTrackToFront -- Bring currTrack to front layer
CancelMessage -- Call from MovieControl handler to cancel controller action
CopyFrame -- Copy current frame to clipboard
CopyPoster -- Copy poster frame to clipboard
FindNextText -- Find text specified by Find properties (the result: time dur offset)
FindNextTextAgain -- Find text again (from last found track, time, offset)
GoNextKeyFrame -- Advance to next key frame
GoPrevKeyFrame -- Go back to previous key frame
GoToBack -- Send window to back
GoToFront -- send window to front
HideController -- Hide the movie controller
HiliteText -- Hilte text specified by Hilite properties above
Idle -- Keeps movie going from inside a script
LoadSegIntoRAM -- Load SegmentStart/SegmentEnd into RAM
PassMouseDown -- Call from MouseDown handler to let controller handle click
PasteBitMapClip -- Set Clip of movie based on bitmap on clipboard
Pause -- Pause the movie
Play -- Start movie playing at current rate
Reverse -- Start movie playing at opposite current rate
SendTrackToBack -- Send currTrack to back layer
SendGroupToBack -- Send all tracks in currTrack's group to back layer
ShowController -- Show the movie controller (works in Direct now)
ShowMovieInfo -- Display Standard QuickTime Movie Info Dialog (QT 2.0)
ShowPoster -- Show the poster frame
StepFwd -- Step Forward one frame (and pause)
StepRev -- Step Backward one frame (and pause)

Get Properties

AbortPlotPathOnClick -- default: true
AudioLevel -- current movie volume
Badge -- default: false
BackColor -- default: white
Cliprect -- the movie's clip rect
CloseOnFinish -- default: false
CurrTime -- current movie time
CurrTrackNum -- default: 0
CurrTrackType -- the type (eg 'soun', 'vide', 'text') of currTrack
CurrTrackLayer -- layer number of currTrack
CurrTrackAudioLevel -- get currTrack's volume
CurrTextSample -- text of currTrack (or first text track) at time set by
                              TextSampleTime (-1: use curr movie time)

CursorMsg -- default: nil
DontPaintWhite -- default: false
Duration -- duration of movie
FastIdle -- default: false
FileSize -- the size of the movie file in bytes
ForeColor -- default: black
HasController -- default: true
Loop -- default: false
MouseDownMsg -- default: nil
MovieController -- actual movie controller handle
MovieControlMsg -- default: nil
MovieHandle -- actual movie handle
MovieLanguage -- the movie's current region code
MovieLanguages -- list of all region codes found in movie's tracks
MovieLoc -- movie's loc within the window
MovieRect -- movie's rect
MovieScale -- the movie's time scale
Mute -- default: false
NumTracks -- number of tracks in this movie
Palindrome -- default: false
PathStartPt -- what you set above
PathEndPt -- what you set above
PathStartTime -- what you set above
PathEndTime -- what you set above
PathNumFrames -- what you set above
PathPlayFrames -- default: false
Rate -- movie's current rate
SeeAllFrames -- default: false
SegmentEnd -- default: -1
SegmentStart -- default: -1
StatusMsg -- default: nil
TimedCallback -- default: nil
VideoCompressorInfo -- list of info for currTrack (or 1st vid track)
Version -- Date XCMD was last compiled
WindowCloseMsg -- default: nil
WindowLoc -- window location (global pt)
Windowname -- the name of the window

WindowRect -- window rect (global)QTEditMovie

Form
EditMovie <fileName>,<windowType>,<loc>[,options...]

Options
NewMovie -- create new empty movie file fileName
Invisible -- hide window initially

Set Properties:

ActiveSegment <beginTime && endTime> -- set the active movie segment
AddToDataFork <true/false> -- make movie DOS compatible in FlattenMovie
ActiveTracksOnly <true/false> -- omit inactive tracks in FlattenMovie command
AutoSave <true/false> -- save changes when edit window is closed
BitMapMovieClip <rect> -- use bit map on card at rect to set movie's clipping region
BitMapTrackClip <rect> -- use bit map at rect to set currTrack's clipping region
CurrGroupLayer <layerNum> -- set layer of all tracks in currTrack's group
CurrSelection <beginTime && endTime> -- set the movie selection
CurrTrackLanguage <region code> -- set language of currTrack
CurrTrackLayer <layerNum> -- set layer of currTrack
CurrTrackQuality <quality> -- set quality value for currTrack
CurrTrackNum <trackNum> -- set the "currTrack" to trackNum
CurrTrackRect <rect> -- set the rectangle of the current track
CurrTime <time> -- set current movie time
DestMovie <name> -- set up file name for FlattenMovie command
DisableTrack <trackNum> -- disable track trackNum
DisplayTracks <true/false> -- show view of tracks below movie
DisplayGroupNums <true/false> -- show group numbers (arbitrary values)
DisplayTrackNums <true/false> -- show track numbers
DontDimController <true/false> -- dont dim movie controller when inactive
DontInterleave <true/false> -- data interleave flag for FlattenMovie command
EnableTrack <trackNum> -- enable track trackNum
FontName <name> -- font to use for AddText/AddSelectedText
FontSize <size> -- font size to use for AddText/AddSelectedText
FontFace <flags> -- face to use for AddText/AddSelectedText
GrabDoneMsg <handler> -- call when audio grab is complete
GroupType <OSType> -- set type for GroupEnabledTypedTracks call
Growable <true/false> -- show movie controller grow icon
HiliteColor <rgb triplet> -- hilite color for AddText calls/AddHilite
Justification <just> -- justfication for AddText calls (0: left, 1:center, -1:right)
Loop <true/false> -- sets loop mode
MovieClipRect <rect> -- set clipping rectangle on movie
MovieLanguage <region code> -- set movie's language
PlayMovieWhileGrabbing <true/false> -- play movie during audio grab
PosterTime <time> -- set the poster time for the movie
PreviewDuration <time dur> -- set the preview duration for the movie
PreviewTime <time> -- set the preview start time for the movie
ScrollDelay <time> -- scroll delay to use for scrolled text
SegmentPlayMode <true/false> -- play only the current selection
SlideTrack <ticks> -- slide currTrack in time by ticks
SlideTrackTime <time> -- slide currTrack by specified time (in movie scale)
SoundDuration <ticks or "movieLength"> -- sets duration of sound to be grabbed
SoundName <name> -- set up resource name for AddSoundResource command
SoundStart <ticks> -- system time to start grab for GrabAudioNow(0: don't wait)
SoundEnd <ticks> -- system time to stop grabbing audio (0: use sound duration)
SoundPlayThru <true/false> -- play sound thru Mac speaker during audio grab
SoundRate <"11K" or "22K" or "44K" or 0> -- set audio grab rate (0: use default)
Stereo <true/false> -- grab stereo sound during audio grab

StopGrabbingOnClick <true/false> -- halt audio grab when mouse clicked
Text <text string> -- set up string for AddText command
TextBackColor <rgb triplet> -- text background color for AddText calls
TextBox <rectangle> -- text box to use for AddText calls
TextFieldName <name> -- HyperCard field to get text from for AddFieldText
TextFlags <flags> -- text display flags to use in AddText calls
TextForeColor <rgb triplet> -- text color for AddText calls
TextHiliteBegin <offset> -- offset for AddText calls/AddHilite (-1 for no hilite)
TextHiliteEnd <offset> -- offset for AddText calls/AddHilite (-1 for no hilite)
TextTrackRect <rect> -- set rect for AddTextTrack command
TrackClipRect <rect> -- set clipping rectangle on currTrack
TrackShiftTicks <ticks> -- number of ticks to slide tracks when stepper clicked
TrackShiftTime <time> -- amount in movie scale time to slide when clicked
UseHiliteColor <true/false> -- use the HiliteColor specified (use default if false)
UseTextBox <true/false> -- use the TextBox specified (use default if false)
Visible <true/false> -- show hide the edit window
WindowCloseMsg <handler> -- call when window closed
WindowName <name> -- set name of edit window

Messages
Add -- Add current clipboard to movie in parallel
AddFieldText -- Add a text sample from field in TextFieldName
AddHilite -- Use text hilite properties to add hilite sample to text track
AddScaled -- Add current clipboard to movie in parallel scaled to selection
AddSelectedText -- Add text sample from currently selected HyperCard text
AddSoundResource -- Add sound track from snd resource (see SoundName)
AddText -- Add a text sample from Text property
AddTextTrack -- Use TextTrackRect property to add new text track
BringGroupToFront -- Bring all tracks in currTrack's group to front layer
BringTrackToFront -- Bring currTrack to front layer
Clear -- Clear the current movie selection
Copy -- Copy current movie selection to clipboard
CopyFramePict -- Copy current frame to clipboard
CopyTrack -- Copy currTrack to the clipboard
CopyTrackSelection -- Copy selected portion of currTrack to clipboard
CutCurrFrame -- Cut current frame from movie
Cut -- Cut current movie selection (copy on clipboard)
CutTrack -- Cut currTrack (copy on clipboard)
CutTrackSelection -- Cut selected portion of currTrack (copy on clipboard)
FlattenMovie -- Create stand alone movie file
GrabAudioSoon -- Prepare to grab audio (must call before GrabAudioNow)
GrabAudioNow -- Grab audio sample to a new sound track
GroupEnabledTypedTracks -- Group all enabled tracks of type GroupType
GroupAllEnabledTracks -- Group all currently enabled tracks
Paste -- Paste current clipboard into movie (insert)
Pause -- Pause the movie
Play -- Play the movie
SaveChanges -- Save changes made so far to movie file

SelectMovieAlternates -- Force Quicktime to select tracks from alternate groups
SendTrackToBack -- Send currTrack to back layer
SendGroupToBack -- Send all tracks in currTrack's group to back layer
Undo -- Undo the last change to the movie (works for most commands!)
UnGroupTracks -- Ungroup all tracks in currTrack's group

Get Properties
AddToDataFork -- default: false
ActiveSegment -- retrieve movie's active segment (startTime && endTime)
ActiveTracksOnly -- default: false
CurrSelection -- The current movie selection (begin && end)
CurrTime -- the current movie time
CurrTrackLanguage -- the language (region code) or currTrack
CurrTrackLayer -- the layer number of currTrack
CurrTrackNum -- number of currently selected track
CurrTrackQuality -- the quality value of curTrack
CurrTrackRect -- the rect of currTrack
CurrTrackType -- the type (eg 'soun', 'vide', 'text') of currTrack
DestMovie -- default: nil
DisplayTracks -- default: false
DontInterleave -- default: false
Duration -- The duration of the movie (in movie's time scale)
FontFace -- default: plain
FontName -- default: App Font
FontSize -- default: 12
GrabDoneMsg -- default: nil
HiliteColor -- default: white
Justification -- default: 0 (left)
MovieChanged -- true if movie has changed
MovieLanguage -- movie's current language
MovieRect -- movie's rectangle
MovieName -- movie's name
MovieScale -- the movie's time scale
NumTracks -- the number of tracks in the movie
PlayMovieWhileGrabbing -- default: false
PosterTime -- the movie's poster time
PreviewDuration -- the movie preview's duration
PreviewTime -- the movie preview's start time
ScrollDelay -- default: 0
SoundDuration -- default: 0
SoundStart -- default: 0
SoundEnd -- default: 0
SoundName -- default: nil
Text -- default: nil
TextBackColor -- default: white
TextFlags -- default: 0
TextForeColor -- default: black
Version -- date XCMD was last compiled

WindowCloseMsg -- default: nil

QTRecordMovie

Forms

QTRecordMovie VideoInputList -- list of digitizer video inputs -> the result
QTRecordMovie CodecList -- list of codecs -> the result
QTRecordMovie DepthList,codecNumber -- list of depths for codec -> the result
QTRecordMovie <windowName>,<windowType>,<windowRect>,
              <growable>,<connectToAudio>,<inputStandard>,
              <video input>[,options...]

WindowType -- standard window types or wdef id#
WindowRect -- rect of window (local coordinates)
Growable -- <true/false> allow drag click in corner to grow window
ConnectToAudio -- <true/false> connect to audio digitizer (if present)
InputStandard -- (ntsc,pal, or secam)
VideoInput -- eg: "Composite 1", "SVideo 2" (Get from VideoInpuList)
Options:
       "Invisible" -- hide window initially

Set Properties

AppendGrab <true/false> -- Append grabbed frames to file (else delete old file)
AudioLevel <0-256> -- audio level for sound play thru
BeepOnGrab <true/false> -- beep when GrabAnotherFrame complete
BlackLevel <0-65535>
Brightness <0-65535>
CodecType <OSType> -- compression type (eg 'rpza', 'jpeg')
CodecNumber <num> -- position from CodecList (instead of codeType)
Contrast <0-65535>
CropRect <rect> -- rect within window to crop the video
CropWindow <true/false> resize, move window to just fit cropped video
DeleteFile <true/false> -- delete prev file before grabbing (except on append)
FileName <name> -- name of output file for grabbed movie
FrameDifferenced <true/false> -- generate video key frames
FrameRate <fps> -- frames per second for resulting movie
GrabCompleteMsg <handler name> call when GrabAnotherFrame complete
GrabSize <point> -- Size at which to grab video (will be saved at window size)
GrabToRAM <true/false> -- Do live grab directly to RAM (else to disk)
Hue <0-65535>
KeepVideoOn <true/false> -- leave video on when HyperCard deactivated
MovieCreator <OSType> -- creator type for output movie file. Default: 'tvod'
PictureQuality <0-1023> -- compression spatial quality
MotionQuality <0-1023> -- compression temporal quality
MovieDepth <depth> -- set depth to grab (1,2,4,8,16,32) (gray: 33,34,36,40)
KeyFrameRate <rate> -- minimum distance between key frames
MaxGrabTime <seconds> -- maximum time (in seconds) for live video grab
MaxGrabTicks <ticks> -- maximum time (in ticks) for live video grab
PictCreator <OSType> -- creator type for file created by GrabPict. Default: 'ppxi'

RestoreSoundPrefs <prefs name> -- Restore saved sound settings from named resource
RestoreVideoPrefs <prefs name> -- Restore saved video settings from named resource
Saturation <0-65535>
Sharpness <0-65535>
SaveSoundPrefs <prefs name> -- Save current sound settings into named resource
SaveVideoPrefs <prefs name> -- Save current video settings into named resource
ShowPrevFrameWindows <true/false> -- show prev two frames grabbed
SoundPlayThruPreview <true/false> -- play sound thru during preview mode
SoundPlayThruRecord <true/false> -- play sound thru during record
SoundRate <"11K" or "22K" or "44K" or 0> -- audio rate for live grab (0: default)
Stereo <true/false> -- capture stereo sound
StopGrabbingOnClick <true/false> -- stop live grab when mouse clicked (default: true)
ThrottleLiveGrab <true/false> -- attempt to limit frame rate on live grabs
VideoCard <input> -- switch to specified digitizer card (from VideoCardList)
VideoInput <input> -- connect to new input (from VideoInputList)
VideoRect <rect> -- adjust digitizer's video rectangle
Visible <true/false> -- show hide video window
WindowCloseMsg <handller name> call when window closed
WindowSize <point> -- new size for video window
WhiteLevel <0-65535>

Messages
AudioOn -- Tuen sound channel's audio on
AudioOff -- Tuen sound channel's audio off
CutCurrFrame -- if PlayMovie was sent then use this to cut unwanted frames
DoLiveGrab -- Start live grab now
FinishControlledGrab -- Close movie started by StartControlledGrab
GrabOneFrame -- Grab frame, compress it, and add it to the movie
GrabPict -- Grab Current image, compress it, and save to Picture file
Idle -- Send if looping in script during record to show frames
LiveGrabPrep -- Get ready to do a live video/audio grab
ReleaseSound -- Disconnect from the audio digitizer
ResetVideoRect -- reset digitizer's video rect to default
ShowController -- During Controlled grab show movie controller to play movie
ShowSoundDialog -- Bring up QT standard audio dialog
ShowVideoDialog -- Bring up QT standard video dialog
StartControlledGrab -- Prepare for calls to GrabOneFrame
StartSound -- Connect to the audio digitizer
UseCropForVideoRect -- set video rect to current cropRect
VideoOn -- unfreeze the video
VideoOff -- freeze the video

Get Properties
BlackLevel -- the current black level setting
Brightness -- the current brightness setting
CodecList -- list of available compressors (by name)
CodecNumber -- Default: codecNumber corresponding to 'rpza'
CodecType -- Default: 'rpza' (Apple Video Codec)

             CodeTypes -- list of available compressors (by type)
             Contrast -- the current contrast setting
             CropRect -- the current crop rect for the video window
             DeleteFile -- default: true
             FileName -- Default: "Temp Movie"
             FrameRate -- Default: 10
             FrameDifferenced -- what you set above
             GrabCompleteMsg -- Default: nil
             GrabSize -- Default: "0,0" -> grab at normal window size
             GrabToRAM -- what you set above
             HighQuality -- the standard high quality value
             Hue -- the current hue setting
             LiveGrabMsg -- Default: nil
             LowQuality -- the standard low quality value
             MaxQuality -- the maximum compresion quality value
             MaxRect -- the maximum video rect you can set for the digitizer
             MinQuality -- the minimum compresion quality value
             MotionQuality -- Default: codec dependent
             MovieDepth -- Default: codec dependent
             NormalQuality -- the standard normal quality value
             PictureQuality -- Default: codec dependent
             Saturation -- the current saturation setting
             Sharpness -- the current sharpness setting
             SoundRate -- Default: Audio digitizer dependent
             ThrottleLiveGrab -- what you set above
             Version -- date XCMD was last compiled
             VideoInputList -- list of video inputs for the digitizer
             VideoCardList -- list of available video video digitizer cards
             Videorect -- the current digitizer video rect
             VideoOn -- whether video is currently on or not
             WindowCloseMsg -- Default: nil
             WindowSize -- the curent size of the video window
             WhiteLevel -- the current white level setting

QTPict

Forms
     QTPict CompressPict -- Compress specified picture
          PictName -- name of pict file/resource
          Source -- "file" or "resource"
          Quality -- spatial quality value (0-1023)
          Codec -- Codec type (eg: 'jpeg', rpza')
     QTPict DisplayPict -- Splat picture onto HyperCard card (rect of pict -> the result)
          PictName -- name of pict file/resource
          Location -- point or rect on card to display picture
          Source -- "file" or "resource"
          "ClipTo",rect -- rect in which to clip image

       "ThumbNail" -- Display thumbnail (for Pict or Movie file)

       "ForceOffscreen" -- Force pict to be buffered offscreen before displaying

QTPict PictBounds -- Return rect of pict in the result (same params as DisplayPict)

QTPict PictRsrcToFile -- convert picture resource to picture file

       RsrcName

       FileName

QTPict PictFileToRsrc -- convert picture file to picture resource

       FileName

       RsrcName

QTPict CodecNames -- Return list of codec names

QTPict CodecTypes -- Return list of available Codecs (by type)

QTPict FileSize -- Return byte size of specified file

       FileName

QTPict ClipTo -- Set clip rect of Hypercard card

       ClipRect

QTPict DiffClip -- subtract rect from clip of Hypercard card

       ClipRect

QTPict UnionClip -- add rect to clip of Hypercard card

       ClipRect

QTPict GetScreenDepth -- Return depth of current screen

QTPict ScreenBitsToPictFile -- grabs whatever is on the screen at rect into pict file

       Rect

       FileName