The QuickTime XCMDs

Ken Doyle
QuickTime Software Group
Apple Computer, Inc.

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. The QTEditMovie XCMD allows you to perform various editing functions including adding and deleting audio tracks, setting a clip region on a movie, and 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.

Displaying QuickTime Movies

There are two methods by which movies can be displayed and controlled. The first method displays movies in external windows called XWindows. XWindows are new to HyperCard 2.0. They allow you to create a window to which HyperCard will send appropriate events such as mouse clicks, update events, and keyboard events. Movies in an XWindow can be controlled either by a graphical controller that appears in the window with the movie or by sending messages to the window. The second method of displaying and controlling movies is to have the movie played directly onto the card window (the new DirectWindow option can override what window direct movies are played in – see below). When a movie is started using this direct to screen method, a movie ID is returned. The movie can then be controlled by subsequent calls to the XCMD with a command name and the movie ID.

To start a new movie, issue the following command in HyperTalk:

```
QTMovie  OpenMovie, windowType,  <fileName>,  location  [,options...]
get the result
if “Error” is in it then <do error handling>
```

To open a movie for playing, the first parameter is “OpenMovie”. The second parameter is either a windowType (specified below) or the keyword “Direct”, which means that the movie will display directly onto the card window, and not in a

separate XWindow. Advanced users can alternatively specify an integer that corresponds to a window definition ID (WDEF). Yet another alternative is to specify MovieWDEF, a custom WDEF for movies (see below). 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 sized 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.

mute	- start the movie in a muted state
paused	- start the movie in a paused state at time zero
closeOnFinish	- close the movie and dispose of all data structures when the movie reaches completion.
showPoster	- shows the poster associated with the movie (implicitly puts movie in pause)
showPreview*	- shows the preview associated with the movie
loop	- when play hits the end, loop back to the beginning (also loops when in reverse)
palindrome	- when play hits the end, play in reverse back to the beginning and repeat
seeAllFrames	- Plays every frame in the movie (no sound)
clipTo,rect	- Only show the portion of the movie corresponding to the rect
loadIntoRAM	- attempts to load the entire movie into RAM for better performance
invisible	- initially, do not show the window (or movie if direct)
FastIdle	- Allows movies to play faster at some sacrifce (see below)

The following options apply only when a Window type has been specified:

noController	- do not display a movie controller in the window
badge	- start with badge mode turned on
dontPaintWhite	- do not paint the background of the window white when it is first drawn (useful when displaying windows on black or colored backgrounds)

The following options apply only when a Direct type has been specified:

DirectWindow,title	- Display movie in specified window rather than the default card widow

All of 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. It will not close when finished. The window will have a controller and it will be visible.

As mentioned above the windowType parameter specifies the window type in which to display the movie (except in the case of Direct). The window types are:

document

windoid

tallWindoid

plain

dialog

altDialog

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

Examples of playing a new movie:

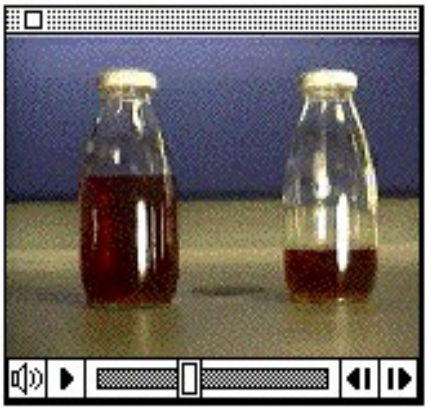
QTMovie OpenMovie, Windoid, “HD:Movies:Running Horses”, “100,100”
-- plays a movie in a windoid located at 100,100 with a movie controller

QTMovie OpenMovie, Direct, “HD:Movies:Tiger”, the rect of btn movie, closeOnFinish
-- plays a direct movie scaled to the size and location of button “movie”. The movie closes on completion (see below about the Idle command to actually get a direct movie to play)

QTMovie OpenMovie, Document, “HD:Movies:Sunset”, “10,10”, paused, invisible, noController
-- starts a movie in an hidden document window, paused at the beginning of the movie, without a controller. The window is made visible by executing “show window Sunset”.

Controlling Movies

Once you have called QTMovie with either a window type or Direct, the method of controlling the two types of movie is quite different. 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.

If you set the EnableKeys property (see setting properties below), 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).

You can also control movies by issuing HyperTalk commands. It is the form of these commands that differs between window movies and direct movies. For a window movie the general form of a command is:

```
send <command> 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.) To tell a window movie to play in reverse (a command not readily available in the controller) you would send the command:

```
send Reverse to window "Running Horses"
```

The list of commands you can send are:

Play	- Play Forward (sets speed to last rate set (see rate property below))
Pause	- Pause (sets speed to 0)
Reverse	- Play Reverse (plays in reverse at last rate set)
StepFwd	- Step Forward 1 "frame"
StepRev	- Step Backward 1 "frame"
CopyFrame	- Copy current frame onto the clipboard
CopyPoster	- Copy poster frame onto the clipboard
HideController	- Hide the play controller (the window is resized)
ShowController	- Show the play controller (the window is resized)

```
LoadSegIntoRAM  - Load the segment into RAM  (see segmentStart and segmentEnd below)
ShowPoster      - Show the Poster frame
PassMouseDown   - Sent from mouseDown call back (see below)
PasteBitMapClip - Set clip of a movie from bitmap on clipboard (see below)
```

For direct movies you first need to have a movie ID for the movie you wish to control. The movie is obtained by getting the result immediately after calling the QTMovie Direct command:

```
global movieID
QTMovie OpenMovie, Direct, “Running Horses”, “100,100”
put the result into movieID
```

Once you have the movieID, you issue commands as follows:

```
QTMovie Direct, movieID, <command>
```

In this case the Direct keyword is first, indicating to the XCMD that what follows is a command to a Direct movie. For example, to tell a direct movie to step forward one frame, you would execute the following script:

```
global movieID
if movieID is not empty then
    QTMovie Direct, movieID, StepFwd
end if
```

The control commands that you can send are: Play, Reverse, Pause, StepFwd, and StepRev. (You can also use the Set command below to control the rate and audio level of the movie.) There are a few other commands not directly related to controlling the movie. The most important by far is *Idle*. When you are running a direct movie, you must issue the Idle command in HyperCard’s idle handler:

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

The Idle command gives the XCMD a chance to keep the movie going. In order for the movie to play at a reasonable speed, Idle should be called as often as possible. (When a movie is running in an XWindow, HyperCard automatically gives the window idle time for it to keep the movie running.) 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.

Another command is Dispose. When you are done with a direct movie, you must call Dispose, so that the movie's data structures are freed:

```
QTMovie Direct, movieID, Dispose
```

To close a movie in a window, use the close window command:

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

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

Getting and Setting Properties of Movies

The QTMovie XCMD supports several Set and Get property calls for movies. The general form of the calls for window movies is:

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

The general form of the calls for direct movies is:

```
QTMovie Direct, movieID, Set, property, value
QTMovie Direct, movieID, Get, property
```

When you use Get with Direct movies the value of the property requested can be found in "the result." The properties that that can be set or retrieved are (note that some apply only to window movies):

windowName	- you can get or rename the window title
currTime	- get or set (in the movie time base) the current time (does **not** pause)
duration	- return (in the movie time base) duration of the movie
audioLevel	- the current audio level
mute	- turns on or off muting (set to true or false)
loop	- turns on or off loop mode (set to true or false)
palindrome	- turns on or off palindrome mode (set to true or false)
rate	- fixed number - sets playback rate of the movie (negative for playing in reverse). If movie was paused, it stays paused. Use send play to start movie going at new rate.
closeOnFinish	- set the close on finish flag (see closeOnFinish option above)
segmentStart	- specify start time for segment play
segmentEnd	- specify end time for segment play
segmentPlay	- set to true or false to turn segment play on or off
fileSize	- return the size in bytes of the movie file
windowCloseMsg	- the name of a HyperCard handler to call when the window is closed
mouseDownMsg	- the name of a HyperCard handler to call when mouse clicked in movie area
timedCallBack	- the name of a HyperCard handler concatenated with a movie time. The
cursorMsg	- the name of a handler to be called when cursor is over the window.
badge	- set badge mode to true or false

seeAllFrames	- set seeAllFrames mode to true or false (see above)
newMovieFile	- specify the filename of a new movie to play in the same window (plays at current rate and volume)
replaceTime	- if you set this before setting newMovieFile or activeMovie, the new movie will start at this time
movieLanguage	- start playing audio tracks with the given country code (if any exist)
movieLanguages	- returns a list of the country codes found in the tracks of this movie
bitMapClip	- Specify a rectangle within your card window. Any bitmap there is used as a template to which the movie will be clipped and shown.
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
eraseOnMove	- erase previous location after a relocation or resize. Direct movies only. Default: true
enableKeys	- enable or disable keyboard control of movie controller
foreColor	- set the foreground color of the movie controller (normally black)
backColor	- set the background color of the movie controller (normally white) The color is expressed as an RGB triplet (where each component is a value between 0 and 65535)
dontInvalOnClose	- useful when closing a window on closeCard. The image of the window is not erased when it is closed. Prevents current card from updating before moving on to next. (set to true or false).

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 palindrome of window “Choir” to “true”
set currTime of window “High Jumper” to field saveTime
set rate of window “Race Car” to 2.5

set foreColor of window foo to “65535,0,0”
set backColor of window foo to “0,65535,0” -- a red on green controller (very Christmassy)

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
QTMovie Direct, movieID, Set, bitMapClip, the rect of btn France
```

If you execute the following statement:

```
answer the properties of window <windowName>
```

you will get a dialog box containing a list of items remarkably similar to: “GeneralProperties1, GeneralProperties2, WindowProperties, DirectProperties, Messages1, Messages2, Version”. You can then ask for the specific list of properties and get another list of specific properties.

```
get GeneralProperties1 of window <windowName> --properties of both window and direct
```

```
movies
get WindowProperties of window <windowName> -- specific to window movies
get Messages1 of window <windowName> -- messages you can send to the XCMD
or
QTMovie Direct, movieID, Get, DirectProperties
get the result
etc.
```

Window and Movie Positioning and Resizing

The initial window position is determined by the position 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 exists). 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.

Finally, the clipRect property can be set on a movie. This 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.

Call Back Messages

Four call back messages are supported by the XCMD: windowCloseMsg, mouseDownMsg, cursorMsg, and timedCallBack. Each cause a message to be sent to the current card. A handler by that name should be defined to run when the call back occurs. The windowCloseMsg is called when the window is about to be closed. It is called with the window name as a parameter. The mouseDownMsg is 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 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 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 msg 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. Examples:

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

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 (windowCloseMsg, cursorMsg, and mouseDownMsg cannot). 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 behavior of clicking in the movie to stop and double clicking to 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
```

Segment Play

The segment play properties allow you to play a segment of a movie:

```
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 sendinmg the loadSegIntoRAM message:

```
send loadSegIntoRAM to window “myMovie”
```

Plotting Frames of a Direct Movie Between Two Points

A special set of properties exist that only apply to Direct movies:

pathStartPt, pathEndPt	the start and end points of a path
pathStartTime, pathEndTime	the start and end times in the movie (Defaults to entire movie)
pathNumFrames	How many frames to plot along the path
pathPlayFrames	Tells whether to play frames at each interval (default: false)

If you set up these properties and then call the PlotPath command 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
        put 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 plotting of the movie is aborted by any OS Event (such as a mouse click or a key press).

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 accomodate 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 queued movies 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 1991 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.

Notes on Using Direct Movies

Disadvantages: Direct movies are blasted directly onto the card window. If HyperCard tries to update, the image will be erased. Direct movies must be disposed of explicitly or you will rapidly run out of memory. You must send idle in an idle handler. You must keep track of the movieID returned by OpenMovie. No standard movie controller. No mouseDown or cursorMsg call backs. No badges.

Advantages: PlotPath; You can use to leave image on screen after movie is disposed; It is simpler to place buttons behind the movie. The new DirectWindow option, which allows one to specify by name an alternate window in which to play the movie, may prove useful in avoiding refresh problems. It should allow some interesting alternate controllers by displaying movies in windows created with HyperCard 2.0's Palette Maker, which is capable of displaying color picts.

The following script will refresh a paused Direct movie:

```
on refesh
    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.

The MovieWDEF windowtype

If you specify 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:

The window can have a controller (which is the default). 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.

```
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
```

This 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 clip region on a movie.

If you set the bitMapClip property of a movie before making the window visible, then you can display rectangular movies in a window the shape of the clip:

```
QTMovie OpenMovie, MovieWDEF, “HD:Movies:NormalMovie”, “10,10”, cmdKeyDraggable,
    noController,invisible
if “Error” is in the result then <error handling>

set bitMapClip of window NormalMovie to the rect of clipButton
show window NormalMovie
```

There is a new command you can send, PasteBitMapClip, which will 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 on the screen would be to have the bitmap hidden behind an opaque button:

```
on ShowFunnyWindow
    set lockScreen to true
    choose select tool
    drag from topLeft of button "shape hider" to botLeft of button "shape hider"
    domenu copy picture
    choose browse tool

    QTMovie OpenMovie, MovieWDEF, “HD:Movies:NormalMovie”, “10,10”, cmdKeyDraggable,
    noController,invisible
    if “Error” is in the result then <error handling>

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

Director and SuperCard Support

The QTMovie XCMD has been tested minimally and has been seen to work in Director 3.0 and SuperCard 1.5. A bug in Director 2.0 (for which there is supposedly a patch) prevented the XCMD from working in that version. You must use Direct movies, as currently neither of these applications support HyperCard 2.1 style XWindows.

The QTRecordMovie XCMD

To record a movie, you must first open a live video window by sending the QTRecordMovie command:

```
QTRecordMovie windowName, windowType, windowRect, growable, previewSound, videoStandard
```

The XCMD will search for a 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 command 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 grow box. If you specify true for the previewSound parameter, then sound will play through (assuming you have an audio digitizer properly hooked up). VideoStandard is an optional parameter that can be “ntsc”, “pal”, or “secam”. Ideally a video digitizer component should detect what kind of signal is being input to the card and should make this transparent to the XCMD. Unfortunately this is not always true at present. Thus the videoStandard parameter is provided so you can force the digitizer to use the specified standard. If you do not specify this parameter, the default signal of the digitizer is used.

There are a number of commands you can send and properties you can set to create a movie. Hence messages to the window fall into two categories:

```
set <property> of window  <window name> to <value>
send <message> to window <window name>
```

The list of properties are:

```
movieName         - specify the file name for the movie to be created - Default: “Temp Movie”
movieDepth        - the movie compression depth - Default: (determined by compressor)
pictureQuality    - image compression quality (0 to 1023). Default: 512
motionQuality     - frame differencing quality.  Default: 512
frameRate         - the rate that captured frames should be played back. Default: 10
frameDifferenced - Whether the movie is frame differenced. Default: false
keyFrameRate      - The key frame rate for frame differenced movies. Default: 10
cropRect          - Sets the rectangle within the video window that will be grabbed
windowSize        - set the height and width of the window (eg: “160,120”)
Hue               -  controls the video digitizer; scale is 0 to 65535
Saturation        -  scale is 0 to 65535
Brightness        -  scale is 0 to 65535
Sharpness         -  scale is 0 to 65535
Contrast          -  scale is 0 to 65535
BlackLevel        -  scale is 0 to 65535
WhiteLevel        -  scale is 0 to 65535
throttleLiveGrab  -  For very fast machines, set to true to enforce maximum frame rate on live
                        grab  (Default: false)
```

grabToRam	- If true, movie is grabbed to RAM rather than disk. (Default: false)
appendGrab	- If true, subsequent BeginSingleGrabMovie appends to existing file rather than creates new one
SoundRate	- for live grab, the rate at which to grab sound. Default is sound digitizer dependent. You can specify “11K”, “22K”, or a fixed number. Zero will return to the default choice.

There are a few properties specific to choosing a compressor:

codecType	- Set the four character name. Default: “rpza”, Apple’s Video compressor.
codecList	- returns a return delimited list of full codec names
codecNumber	- You can set the codec by number, which is its position in the codec list

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, highQuality

The list of commands you can send are:

LiveGrabPrep	- get ready to grab a live movie
DoLiveGrab	- Start Live grab now
BeginSingleGrabMovie	- Prepare for grabbing one frame at a time
GrabOneFrame	- Grab one frame and add it to the current movie
EndSingleGrabMovie	- Close the movie
VideoOn	- Unfreeze the Live video window
VideoOff	- Freeze the Live video window
AudioOn	- Unmute the audio playthru
AudioOff	- Mute the audio playthru
ReleaseSound	- Release the sound channel being used to preview sound
StartSound	- Reallocate the sound channel for previewing sound

If you need to change any of the default properties given, you can simply issue the Set command, for example:

```
Set movieName of window “Live Video” to “MyDisk:Movie folder:Dancing Bear”
Set frameRate of window “Live Video” to 15
Set frameDifferenced of window “Live Video” to false
Set codecType of window “Live Video” to “jpeg”
```

Once you have the properties the way you want them, there are two basic ways to create a movie. One way is to send the LiveGrabPrep and DoLiveGrab commands. This grabs frames as fast as it can (which for now isn’t very fast) into the movie file. You can specify a maximum time via the maxGrabTime property, or click the mouse to stop recording. You can get better results if you grab to RAM using the grabToRam option, but the size will be limited to available RAM space.

The other way to record a movie is to use the sequence BeginSingleGrabMovie, GrabOneFrame (a number of times), and then EndSingleGrabMovie. For example, if

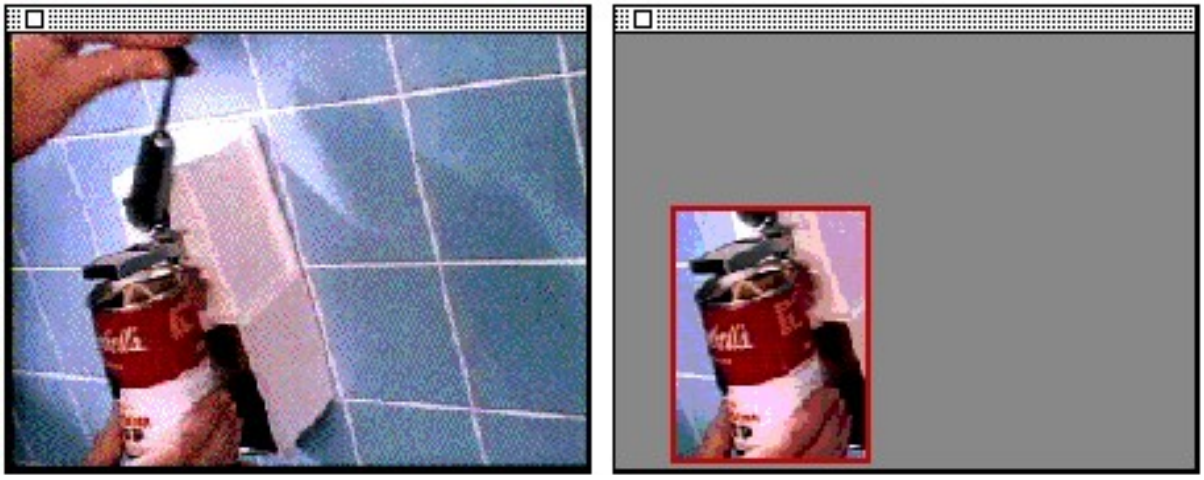
you had XCMDs that controlled a laser disk, you could record frames from it by executing the following script:

```
on GrabSomeFrames startFrame, endFrame
	vidSearch startFrame
	send BeginSingleGrabMovie 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 EndSingleGrabMovie to window “Live Video”
end GrabSomeFrames
```

This would actually grab every frame. Currently, since playback, even at small sizes, rarely gets above 15 frames per second, you may want to grab every other frame to save space. Whatever rate you decide to grab at 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. The keyFrameRate property tells the XCMD how often to generate key frames if the movie is frame differenced. The compressor may decide to insert key frames automatically at scene transitions in addition to the key frame rate specified.

You can also 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 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.



You can also set the cropping rectangle by setting the cropRect property of the window:

```
Set cropRect of window “Live Video” to “0,30,160,100”
```

Special commands for Stop Frame movie making

If you send PlayMovie 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 EndSingleGrabMovie, you can still add to the movie by setting the doAppend property to true before again calling BeginSingleGrabMovie. If doAppend is false, BeginSingleGrabMovie, will replace the file. See the Stop Frame card in the Movie Making Stack.

Note on Window and cropping rectangle location and size

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. The size of video to be grabbed is enforced to be a multiple of four in the horizontal and vertical dimensions. This allows compression and decompression to work much faster. 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 and size may

not show up exactly as you specified.

The QTEditMovie XCMD

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

The QTMovie XCMD described above never modifies the movie file. QTEditMovie is provided to perform a variety of movie editing functions including adding new sound tracks either by capturing from a digitizer, copying from a sound resource, or copying from another sound track. It also supports deleting tracks, time shifting tracks, and grouping and setting languages on audio tracks. You can set a clipping region on a whole movie or on individual video tracks. Tracks can be displayed in a simplified layout in the edit window. You can select individual tracks or multiple tracks. You can also “flatten” a file, such that unused data is removed and sound data is interleaved with video data for better playback performance. You can also use QTEditMovie to set the Poster frame and the movie Preview.

When the EditWindow first opens it looks much like the QTMovie window with the movie controller hanging below the movie (you don’t have an option whether to show it or not). The name of the window is that of the movie file.

Adding Live Audio to a Movie

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 RecordMovie XCMD was
                                                    previewing sound
send GrabAudioSoon to window “My Movie”

--Get ready to grab
send GrabAudioNow to window “My Movie”
```

The soundStart property defaults to zero, in which case the sound grabbing starts immediately after you send grabAudio now. 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”, 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 you favorite video using this method. You should turn the sound down before you do this so that the sound of exisiting tracks does not interfere with your recording.

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

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

Adding a Sound Resource to a Movie

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

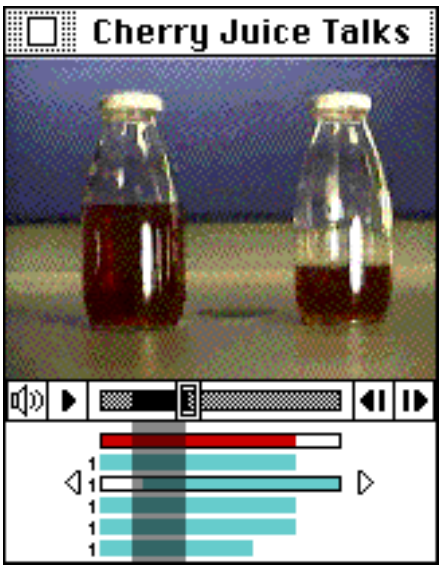
```
set soundName of window “My Movie” to ribbit
send AddSoundResource to window “My Movie”
```

Currently, the sound resource needs to be in the current resource path (such as in the current stack). Support for specifying separate sound files will be added.

Displaying Tracks

You can see a representation of the tracks in a movie by setting the DisplayTracks property:

```
set DisplayTracks of window “My Movie” to true
```



This will give you a view of the various tracks contained in the movie. Video tracks are red, audio tracks are blue, and other types are gray. Enabled tracks are indicated by a black outline. Clicking on a particular track toggles whether it is enabled. If you double click on a track it becomes the “current” track. This is indicated by small stepper buttons that will appear to either side of the current track. By clicking on these steppers you can “slide” the tracks in that direction, thus allowing you to adjust tracks that are out of sync. If you hold down the mouse button over the stepper the track will continue sliding. The amount shifted with each “step” is determined by the trackShiftTicks property. The value of this property is how many ticks (60ths of a second) the track is shifted each time. The default for this property is 6 which equals 1/10th of a second.

Many of the editing operations described below operate on the current track. You can set the current track from a script by setting the currTrackNum property. You can enable and disable tracks by setting the enableTrack and disableTrack properties.

```
set enableTrack of window “My Movie” to 3 -- enable third track
```

If some of the tracks in the movie have been grouped into alternate sets, such as for multiple language tracks, the grouped tracks are indicated by small digits to the left of the track. Each member of the same group has the same number.

To set the language of the current track, set the currTrackLanguage property to the appropriate region code. A list of region codes appears on page 14-133 and 14-134 of Inside Mac, vol 6.

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". Future track types will have their own four character identifier.

To group a set of alternate language tracks, select only the audio tracks you wish to group (you do not need to deselect any video tracks) and then send the GroupSelectedAudioTracks command. You can also group video tracks with the GroupSelectedVideoTracks command. If you want to group any combination of tracks, use the GroupAllSelectedTracks command. You can ungroup by selecting tracks and sending the UngroupSelectedTracks command. Once you have selected languages and grouped tracks, the set movieLanguage property will automatically select the specified language and deselect the other tracks in the same group. This property is not saved with the movie. Movies always start out with the language of the System Software the Macintosh was booted on.

Setting Clip Regions on Movies

You can set a permanent clip on a movie by setting the BitMapClip property. This is similar to the BitMapClip property described for the QTMovie XCMD. You pass a rectangle that designates where on the HyperCard card the bitmap you wish to use is located. The bitmap will be converted to a region and then applied to the

movie

(and will be saved with the movie, unlike QTMovie). You can also set the BitMapTrackClip property which will set the clip only for the currently selected video track. If you zero instead of a rectangle, the clip region will be cleared.

Note that there can be no “obstruction” covering the specified rectangle, such as a corner of another window. If there was, the corner of the window would become part of the clip region.

When the resulting movie is played back with QTMovie using the MovieWDEF window type, the window will be the shape of the clip region you set (see above).

Editing Movies

There are numerous commands to edit the movie window:

```
Cut        - Cut the current selection within all tracks of the movie (puts a movie on clipboard)
CutCurrentFrame        - Cut the current frame within all tracks of the movie (puts a movie on
                                    clipboard)
Copy      - Copy the current selection within all tracks of the movie (puts a movie on clipboard)
Clear                - Clear the current selection within all tracks of the movie
Paste                - Insert the movie on the clipboard into the movie at the current selection
PasteNewTracks    - Add the movie on the clipboard onto a new track starting at time 0
CutTrack             - Cut the entire current track out of the movie (puts a movie on clipboard)
CutTrackSelection - Cut selected part of current track (puts a movie on clipboard)
CopyTrack            - Copy the entire current track (puts a movie on clipboard)
CopyTrackSelection       - Copy selected part of current track (puts a movie on clipboard)
Undo                 - Actually works in most cases including the commands in the above sections.
                            Undo twice will redo
```

If the editing window is active, the cmd/x,c,v,z keys on the keyboard will do the Cut, Copy, Paste, and Undo cammands described above. Shift/cmd/v will do the PasteNewTracks command.

Other Properties

```
CurrTime             - set the time of the movie
PosterTime           - set/get the poster time of the movie
movieRect            - get the rect of the movie
segmentStart         - specify start time for segment play
segmentEnd           - specify end time for segment play
segmentPlayMode - set to true or false to turn segment play on or off
```

Saving Changes

None of the editing operations described above are permanent until you send SaveChanges to the window. You can also set the autoSave property of the window to true and the changes will be saved automatically when you close the window. The movieChanged property will return true if there has been any change to the movie. (Just selecting and deselecting tracks may constitute a change.) A call back

message is provided when the window is closed (as in QTMovie). This can be used to give users a chance to save changes if they have not done so. Set the windowCloseMsg to the name of a handler in your stack. For example if you specified a handler called “WCLose” you might do the following.

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

Flattening a Movie

To flatten a movie is to make a copy of the movie that is a self contained movie in that all of the data referred to by the movie is within the file. Sending the flattenMovie command will flatten the current movie file. This will remove data from deleted tracks. You can specify which file to put 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.

The QTPict XCMD

The QTPict XCMD performs a variety of Pict 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 only two options at this point are *thumbnail* and *clipTo*. 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.

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 codeNameList, codeTypeList, 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 10. The default is 5. Codec is a four character compression type, which can be obtained from 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.]

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.

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 and time of when the XCMD was built. For QTMovie, QTRecordMovie, and QTEditMovie, you can also ask for the version property of the window.

```
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”
```