



VRnacular scripting Information

version 1.1

VRHotwires includes a strong and flexible scripting language that will allow you to customize your project in a myriad of ways...

In general the format for the minimum script is

□

On Trigger target□(#)

□□□□□DoAction parameters

end

for example:

□

On MouseDown Hotspot 9

□□□□□□□□QTVRSetPanAngle 44

end

□

□

so for a sprite it would be

On MouseDown Sprite 1

QTVRSetPanAngle 99

end□

□

This document is updated as of September 2002, so supports all qt6 actions and operands...

In general VRNacular is based exactly on apple's wired code, so we 'stay out of the way' between you and apple's engineers. If the c header says kActionMovieSetRate then we expose a call: MovieSetRate that uses the exact parameters of apple's code. To add synonyms and interpretations, check out the synonyms and macros document...

Quickstart notes for experts:

□

In general VRNacular is like any other wired language. You use triggers and actions, like

On Mousedown Hotspot 9

```
QTVRSetPanAngle 99
```

```
end
```

□

All the triggers and actions show up in a list next to the scripting window.

There are examples of each action in use there also.

If you need more detailed info on that action consult this document.

All action names, operands, and operators are direct mutations of names in Movies.h a universal header file available at apple. If new versions come out with new calls you can try them in vrhotwires by taking kAction off of the actions, and for operands replacing kOperand with 'the'...

the main 'gotcha's' to watch out for in VRHotwires:

□

1. `QTVRSetPanAngle` expressions can't have spaces.

so if you say

```
QTVRSetPanAngle (theMovieTime+1)/4.7
```

you're ok but if you say:

□

```
QTVRSetPanAngle (theMovieTime + 1)/ 4.7
```

□

you may run into issues.

In 1.1 we wrote a preparser that takes the spaces out of any expression in brackets, but we haven't really had time to test it extensively..

□

2. negative numbers that stand alone need ot use 'neg'

```
QTVRSetPanAngle neg5
```

will work and

QTVRSetPanAngle -5

□

may also work, but if you find a situation where a - is not working, try a neg instead.

also

QTVRSetPanAngle thePanAngle-5

is fine

3. operands that stand alone sometimes need brackets

in some cases, operands that stand alone need brackets.

This is a little odd and may be considered a bug...

You can say

SetPanAngle variable[123]

but you can't say

spriteindex=variable[123]

□

when it's a target you have to say:

□

spriteindex=(variable[123])

□

4. Elseif 1

□

You'll notice that if you put in a script like:

```
if variable[123]=1
```

```
#####QTVRSetPanAngle 99
```

```
else
```

```
#####QTVRSetPanAngle 147
```

```
endif
```

□

the script you'll get back is:

```
if variable[123]=1
```

```
#####QTVRSetPanAngle 99
```

```
elseif 1
```

```
#####QTVRSetPanAngle 147
```

```
endif
```

□

this is just a little eccentricity of qt conditional code. It means the same thing. □ For those

delta makes the parameter work relatively rather than absolutely... so moving a sprite by 5,5, delta will move it 5 from where you are, wherever you are.

the 'wraps' parameter is used with a min and max value to create a loop.
so if we want a sprite to go to one edge of the screen and wrap around we can use wraps with
min and max set to the edge of the screen...

-
-
-

delta
wraps
toggle
min=
max=

OPERATORS

+ (Add)
- (Subtract)
* (Multiply)
/ (Divide)
| (Or)
& (And)
! (Not)
< (LessThan)
<= (LessThanEqualTo)
= (EqualTo)
!= (NotEqualTo)
> (GreaterThan)
>= (GreaterThanEqualTo)
% (Modulo)
div (Integer Divide)
abs (AbsoluteValue)
neg (Negate)

TRIGGERS

?? is the hotspots or sprite number....

??MouseDown Hotspot ?
??MouseDown Sprite ?
??MouseUp Hotspot ?
??MouseUp Sprite ?
??MouseEnter Hotspot ?
??MouseEnter Sprite ?
??MouseExit Hotspot ?
??MouseExit Sprite ?

□ **FrameLoaded** (goes at the sample level)
NodeEntry (What we call frameloaded events in VR)
□ **Idle**
Idle Sprite ?

For more info on wired actions check apple's doc at
<http://developer.apple.com/techpubs/quicktime/qtdevdocs/PDF/WiredMoviesSprites.pdf>
page 104
□

ALL THE ACTIONS:

□

Movie actions:

MovieSetVolume volume

volume is a number between 1 and 256 or an expression.

examples:

MovieSetVolume 123

MovieSetVolume thePanAngle+123

SetMovieVolume 1

SetMovieVolume 1 delta wrap min=11 max=42

SetMovieVolume theMovieVolume+thePanAngle

□

SetMovieVolume is a synonym that is accepted as valid input also...)

This action sets the global volume of all sound tracks in a movie.

Some people get mixed up between MovieSetVolume and TrackSetVolume...

MovieSetVolume changes the volume in the same way as the volume slider in the movie controller. So one is 'global' to the movie, and another is local to a track...

□

"Supported Flags: IsDelta, WrapsAround

Param1: [short volume]

Default Min: 0, Default Max: 255

This sets the movie's volume level."

MovieSetRate rate

rate is a number or expression.

rate=1 the movie is playing normally.

rate=0 the movie is stopped

rate=-1 the movie is playing backwards.

rate=2 the movie is playing twice as fast as normal.

examples:

```
MovieSetRate 1
```

```
MovieSetRate thePanAngle+123
```

The script to stop a linear movie from playing is

```
SetMovieRate 0
```

(SetMovieRate is a synonym that is also accepted as valid input)

to start it again use:

```
SetMovieRate 1
```

"Supported Flags: IsDelta, WrapsAround

Param1: [Fixed rate]

Default Min: minFixed, Default Max: maxFixed

This sets the movie's playback rate. A rate of 1 means play back at normal speed. A rate of two means play back at double speed. A rate of 0 means stop. Negative rates make the movie play backwards. Rates may be fractional."

MovieSetLoopingFlags flagtype

flagtype is a number or an expression.

flagtype=1 loop the movie(loopTimeBase)

flagtype=0 no loop

flagtype=2 loop is palindrome(palindromeLoopTimeBase)

examples:

```
MovieSetLoopingFlags 1
```

```
MovieSetLoopingFlags thePanAngle+123
```

"Supported Flags: none

Param1: [long loopingFlags]

This sets the looping mode of movie playback. Zero means no looping. Setting the loopTimeBase flag means that the movie will loop; additionally setting the palindromeLoopTimeBase flag causes the movie to loop in palindrome fashion, meaning that once it reaches the end, it goes backwards until reaching the beginning, at which point it will go forward again. Note that the flags loopTimeBase and palindromeLoopTimeBase are OR-combined together."

□

MovieGoToTime time

time is a number or an expression.

time=0 the beginning of the linear movie

time=3600 6 seconds in, depending on a timescale of 600

examples:

MovieGoToTime 123

MovieGoToTime thePanAngle+123

"Supported Flags: none

Param1: [TimeValue time]

This sets the movie's current time. This value is expressed in the movie's timescale."

MovieGoToTimeByName

time is a string (or an expression. ???)

time="start" the beginning of the linear movie

time="jump" 3600 6 seconds in, depending on timescale.

examples:

MovieGoToTimeByName "intro"

To use this call you have to have a linear movie with chapters defined. It will jump you to the chapter of your choice...

"Supported Flags: none

Param1: [Str255 timeName]

Sets the movie's current time to the one in the movie corresponding to the chapter named timeName."

MovieGoToBeginning

no parameters

Takes you to the beginning of the movie.

examples:

`MovieGoToBeginning`

"Supported Flags: none

No Params

Sets the time to the beginning of the movie."

MovieGoToEnd

no parameters

Takes you to the end of the movie.

examples:

`MovieGoToEnd`

"Supported Flags: none

No Params

Sets the time to the end of the movie."

MovieStepForward

no parameters

Takes you one frame forward in the movie

examples:

`MovieStepForward`

"Supported Flags: none

No Params

This causes the movie to step forward in the same fashion as pressing the step forward button in the movie controller."

MovieStepBackward

no parameters

Takes you one frame backwards in the movie...

□

example:

MovieStepBackward

"Supported Flags: none

No Params

This causes the Movie to step backward in the same fashion as pressing the step backward button in the movie controller."

MovieSetSelection startTime endTime

Selects a range of time in the movie like when the movie controller has a dark region because it's a selected time range...

startTime is a number or an expression.

endTime is a number or an expression.

examples:

MovieSetSelection 1 123

MovieSetSelection thePanAngle+4 12345

□

"Supported Flags: none

Param1: [TimeValue startTime]

Param2: [TimeValue endTime]

Sets the movie's selection to be the time range specified by the startTime and endTime time values."

MovieSetSelectionByName

□

Same as above but with a name as the target...

examples:

MovieSetSelectionByName "intro" "chapter 1"

□

"Supported Flags:none

Param1: [Str255 startTimeName]

Param2: [Str255 endTimeName]
Sets the movie's selection to be the time range specified by the startTimeName and endTimeName chapter names."

MoviePlaySelection flag

flag=0 play the selection and keep going.
flag=1 play the selection and loop it.

□

examples:

□□□□MoviePlaySelection 1

So in general, you set a selection and then play it like

MovieSetSelection 100 1000
MoviePlaySelection 1

In VR this is handy as you can use it to loop a sound or video sequence going on at the same time as your pano. For example my 'walker' demo with the person walking around, is a series of movie selections for each body movement...

"Supported Flags:IsToggle

Param1: [Boolean selectionOnly]

When set to true, the movie plays only the current movie selection. The movie selection may be set using kActionMovieSetSelection or kActionMovieSetSelectionByName."

□

MovieSetLanguage language

language is a number or a string like french

examples:

□□□□MovieSetLanguage french

□

□

"Supported Flags:none

Param1: [long language]

Sets the movie's current language. This causes the tracks associated with that language to be enabled and track's associated with other languages to be disabled."

□

" Language numbers are as follows:
english = 0 french = 1
german = 2 italian = 3
dutch = 4 swedish = 5
spanish = 6 danish = 7
portuguese = 8 norwegian = 9
hebrew = 10 japanese = 11
arabic = 12 finnish = 13
greek = 14 icelandic = 15
maltese = 16 turkish = 17
croatian = 18 tradChinese = 19
urdu = 20 hindi = 21
thai = 22 korean = 23"

MovieChanged

no params

example:

```
On MouseDown Hotspot 9
MovieChanged
end
□
□
```

"Indicates that movie has changed in some substantial way. For example:
enabling/disabling a track.

This was actually added for the streaming media handler; it basically bumps the movie-changed seed, letting the toolbox know the movie has been edited. Certain operations such as setting a track's matrix, clip, volume, or enabled state normally bump this seed. In the Wired Actions, we choose not to bump it to avoid a "Do you want to save changes to this movie" type of dialog from QuickTimePlayer. If you want this dialog to appear after using these actions, call `MovieChanged`'

MovieRestartAtTime time rate
□

example:

```
MovieRestartAtTime 36 1.2
□
```

Restart the movie at the specified time at the specified rate...

"This action restarts the targeted movie at the specified movie time, restarting it at the specified rate. If rate is set to zero, then the current movie rate is used. More specifically, this action stops the current movie, changes the movie's time to the specified time, and then prerolls the movie from that time at the specified rate.

"

The following actions go to the time associated to a chapter. The chapter is specified relative to the current chapter or by index.

MovieGotoNextChapter

Changes the movie time to the start of the next chapter.

MovieGotoPreviousChapter

Changes the movie time to the start of the previous chapter.

MovieGotoFirstChapter

Changes the movie time to the start of the first chapter.

MovieGotoLastChapter

Changes the movie time to the start of the last chapter.

MovieGotoChapterByIndex

Changes the movie time to the start of the nth chapter

MovieSetScale x, y

set the current scale of the movie

the values are ratios, so 1.0 and 1.0 will give you the exact scale of the movie

So saying `MovieSetScale 2.0 2.0` will make the movie twice it's normal size.

Track Actions:

!
□

TrackSetVolume volume tracktarget

examples:

TrackSetVolume 123 TrackIndex=5
 TrackSetVolume thePanAngle+theMovieTime TrackIndex=5
□

This call sets the volume of a specific track. So if you have a movie with multiple sound tracks you can target just a single sound. The directional sound tool, for example uses this to allow you to add multiple directional sounds...

"Supported Flags: IsDelta, WrapsAround
Param1: [short volume] Default Min: 0, Default Max: 255
Sets the track's volume level."
□

TrackSetBalance balance tracktarget

eg: TrackSetBalance 123 TrackIndex=5

"Supported Flags: IsDelta, WrapsAround
Param1: [short volume] Default Min: -128, Default Max: 128
Sets the track's left to right balance.
The range of numbers is as follows: -128 = only left;
128 = only right; 0 = even amounts of left and right."
□

TrackSetEnabled enabledstatetracktarget

eg: TrackSetEnabled true TrackIndex=5

"Supported Flags: IsToggle
Param1: [Boolean enabled]
Enables or disables a track."
□

TrackSetMatrix

eg: TrackSetMatrix 0 0 0 1 0 0 0 1 TrackIndex=2
□

"Supported Flags: IsDelta, WrapsAround

Param1: [MatrixRecord matrix]

For each matrix element, the default maximum and minimum values are the largest and smallest possible Fixed values.

Sets the target track's matrix, allowing you to move, resize, rotate, and otherwise distort a track's shape. If you set the Track's matrix to a value that makes its display bounds fall outside of the movie's bounds, then the movie's bounds are resized to accommodate the new track box. Although this works fine in the QuickTime Movie Player, be warned that not all applications support this type of dynamic movie resizing cleanly."

TrackSetLayer

eg: TrackSetLayer 1 1 TrackIndex=5
□

"Supported Flags: IsDelta, WrapsAround

Param1: [short layer]

Sets the target track's layer number. This is used to specify its front-to-back order relative to the other spatial tracks in the movie. The smaller the layer number, the more forward the track appears."

TrackSetClip

□

"Supported Flags: none

Param1: [RgnHandle clip]

Sets the track's clipping region. This parameter contains QuickDraw Region data."

TrackSetCursor

eg: TrackSetCursor 1 28 TrackIndex=5

TrackSetCursor 128 TrackIndex=5
□

"Supported Flags: none
Parm1: [QTAtomID cursorID]
This sets the cursor to the one specified by cursorID. Note that the target is a track. This is because tracks may supply their own custom cursors.
You set the cursor ID as follows:
cursorID = 0
Set the cursor to the system default cursor.
cursorID (1..1000)
You set the cursor to a built-in QuickTime cursor. You use values from this enumeration in Movies.h .

```
enum {  
kQTCursorOpenHand = 128,  
kQTCursorClosedHand = 129,  
kQTCursorPointingHand = 130,  
kQTCursorRightArrow = 131,  
kQTCursorLeftArrow = 132,  
kQTCursorDownArrow = 133,  
kQTCursorUpArrow = 134  
};  
cursorID > 1000  
You set the cursor to one included in the movie.  
Specifically, it is included in the target track's  
MediaPropertiesAtom atomdata. This is an atom of type  
'crsr' with an atom ID corresponding to the desired cursor  
ID (greater than 1000). The atom's data is simply a  
Macintosh 'crsr' resource. On Windows, a  
black-and-white version is currently used."  
??????  
"standard = 0  
downArrow = -19178 closedHand = -19182  
leftArrow = -19179 IBeam = -19176  
rightArrow = -19180 openHand = -19183  
upArrowCursor = -19177 pointingHand = -19181"
```

TrackSetGraphicsMode

eg: TrackSetGraphicsMode TrackIndex=5□{260,{0,0,0}}

"Supported Flags: none
Parm1: [ModifierTrackGraphicsModeRecord graphicsMode]"

Sets the target sprite's graphics mode. Here's a list of mode constants you can use.
//the 16 transfer modes□□□□□□// Arithmetic transfer modes
srcCopy 0□□□□□□□□□□□□□□hilite 50

Sprite Actions:

SpriteSetMatrix

eg: SpriteSetMatrix 1 0 0 0 1 0 23 24 1 TrackIndex=5 SpriteIndex=1

Matrices have a bit of a learning curve associated with them, but they are a powerful way to move things around if learned. In wired quicktime you cannot use variables as elements of a matrix, so you might want to use SpriteTranslate or SpriteScale in situations where you need to hand in a variable...

To set a matrix to do nothing you hand the 'identity matrix'....

```
1 0 0
0 1 0
0 0 1
```

To move the sprite around, you hand in values for the 7th and 8th parameters.
So

```
1 0 0
0 1 0
20 20 1
```

will move a sprite to 20 20

and to scale it you change the numbers in the 1st and 5th position...

```
1.5 0 0
0 1.5 0
0 0 1
```

to ratios you want scale by.

You can also rotate and skew the sprite by changing various values...

"Supported Flags: IsDelta, WrapsAround

Param1: [MatrixRecord matrix]

Sets the target sprite's matrix, allowing you to move, resize, rotate, and otherwise distort a sprite's shape. Sprites are

clipped by their containing sprite track's bounds and clip region."

SpriteSetImageIndex index TargetTrack TargetSprite
□

eg: SpriteSetImageIndex □ 2 TrackIndex=2 SpriteIndex=1
□

A sprite track can have x amount of images in it. Let's say we have a sprite track with 5 images. The first image is on the first sprite, but we want to change it, so it shows the 5th image for a while... Then we say

SpriteSetImageIndex □ 5 SpriteIndex=1

This gets really interesting when one is using the sprite over-ride capability in qt. So Let's say we have a sprite track with 2 sprites, and each of them is playing a video track on top. By changing the index of the sprite, we can map the video track from sprite 2 onto sprite 1. This is handy if you want to play a sequence of short videos on a sprite, you can do so by changing the image index in a sequence.

"Supported Flags: IsDelta, WrapsAround
Param1: [short imageIndex] Default Min: 1, Default Max:
num images

Sets the target sprite's image index. Each sprite track keyFrame contains a list of images. Setting a sprite's image index selects which image from this list is currently displayed. The image index ranges from 1 to the number of images."
□

SpriteSetVisible

eg: SpriteSetVisible 0 TrackIndex=2 SpriteIndex=1

This will set the current sprite's visibility...

"Supported Flags: isToggle
Param1: [short visible]
Shows or hides the target Sprite."

SpriteSetLayer

eg: SpriteSetLayer 3 TrackIndex=2 SpriteIndex=1

This sets the sprite's layer within the track. Not to be confused with TrackSetLayer which will move tracks forward or back by layer...

"Supported Flags:IsDelta, WrapsAround

Param1: [short layer]

Sets the target sprite's layer number. This is used to specify its front-to-back order relative to the other sprites in the sprite track. The smaller the layer number, the more forward the sprite appears. Note that 32767 indicates that this is a background sprite."

SpriteSetGraphicsMode

eg: SpriteSetGraphicsMode {260,{0,0,0}} Sprite="note"

This will allow you to set an individual sprites graphics mode...

"Supported Flags:IsDelta

Param1: [ModifierTrackGraphicsModeRecord graphicsMode]"

SpritePassMouseToCodec

"Supported Flags:none

No Params

Passes the location of the mouse event to the codec that is drawing the sprite's current image. (Note that currently, only the Ripple codec accepts clicks, causing a ripple effect originating from the location).

It only makes sense to use this action in response to mouse-related events."

eg: SpritePassMouseToCodec TrackIndex=2 SpriteIndex=1

SpriteClickOnCodec

Supported Flags:none

Param1: [Point localLoc]

Passes the point localLoc to the codec that is drawing the sprite's current image. (Note that currently, only the Ripple codec accepts clicks, causing a ripple effect originating from the location.)

This action is similar to kActionPassMouseToCodec except the location to click on is a parameter, so it may be used in response to any type of event.

SpriteTranslate

eg: SpriteTranslate 5 5 false TrackIndex=5 SpriteIndex=1

"Supported Flags:none

Param1: [Fixed x]

Param2: [Fixed y]

Param3: [Boolean isAbsolute]

If the isAbsolute parameter is true, this moves the sprite to the absolute location specified by the x and y parameters; if the isAbsolute parameter is false, it specifies how far from the current location to move the sprite. The coordinate system for x and y is the sprite track's source space."

□

□

SpriteScale

eg: SpriteScale 1.2 1.2 TrackIndex=5 SpriteIndex=1

"Supported Flags: none

Param1: [Fixed xScale]

Param2: [Fixed yScale]

Scales the target sprite by xScale and yScale about its current image's registration point. For example, to double a sprite's width and half its height, you would set xScale to two and yScale to one-half."

SpriteRotate

eg: SpriteRotate 5 TrackIndex=5 SpriteIndex=1

"Supported Flags: none
Param1: [Fixed degrees]
Rotates the target sprite about its current image's registration point. The amount of rotation is specified by degrees."

SpriteStretch

eg:SpriteStretch□□ 0 200 0 200 200 0 200□TrackIndex=5 SpriteIndex=1

□

This call takes the four corners of a rectangle.
so 0 0 200 0 200 200 0 200 describes these corners in xy pairs.

You can then hand it variables to skew the rectangle in all manner of ways...

"Supported Flags: none

The eight parameters specify four corners of a four-sided polygon into which the sprite's image is stretched. The coordinate system for points is the sprite track's source space."

□

□

□

SpriteSetCanBeHitTested canBeHitTested.

A new qt6 call...to have "full control over how the sprites in a sprite track interact with mouse clicks."

"When a sprite is clicked in a sprite track, it "receives" the mouse click. However, there are times when you may want to have sprites that do *not* receive a mouse click, and instead, you want the mouse click to "pass through" that sprite (and on to another sprite or perhaps another track behind the sprite track). In earlier versions of QuickTime, this was not possible.

In QuickTime 6, however, this behavior—passing a mouse click through a sprite—can be controlled..."

Sets the value of the hit testing property.

QTVR Actions

QTVRSetPanAngle apanAngle

This is a very fundamental call to qtvr. It allows you to add wiring that Pans the qtvr to the chosen angle...

eg: **QTVRSetPanAngle** 99

□

will take you to pan angle 99.

The flags get used quite a lot with this call for example to pan a movie continuously to the left you might say:

On idle

```
QTVRSetPanAngle -3 delta wrap min=0 max=360
```

end□

"Supported Flags: IsDelta, WrapsAround
Param1: [float panAngle]
Default Min: min Pan Allowed By Media,
Default Max: max Pan Allowed By Media
Sets the target QuickTime VR track's pan angle."

QTVRSetTiltAngle

"Supported Flags: IsDelta, WrapsAround
Param1: [float panAngle]
Default Min: min Pan Allowed By Media,
Default Max: max Pan Allowed By Media
Sets the target QuickTime VR track's pan angle."

QTVRSetFieldOfView

"Supported Flags: IsDelta, WrapsAround
Param1: [float panAngle]
Default Min: min Pan Allowed By Media,
Default Max: max Pan Allowed By Media
Sets the target QuickTime VR track's pan angle."

QTVRShowDefaultView

when you create a pano in an authoring tool, you set a default view.
Calling this action will take you back there.

It has no parameters.

QTVRGoToNodeID aNodeID

eg: QTVRGoToNodeID 3

For the most part navigation in multinodes is done by link hotspots, which is a no wiring affair...

However, in some cases, such as transitions, you want to be able to do something between the time the hotspot is clicked and the time you change nodes... so you need to be able to take over the Node Changing.

Another time it's nice to use this is when you have a hotspot that goes to different nodes depending on a condition. Say to unlock a door in a pano, you need to pull a switch... When the switch is pulled you set variable 1 to 1, and otherwise it 's zero.

```
On MouseDown Hotspot 3
  If variable[1,tracktype=sprite]=1
    QTVRGoToNodeID 3
  else if 1
    QTVRGoToNodeID 2
  endif
end
```

QTVREnableHotSpot ID

Enables or disables a QuickTime VR hot spot by ID.

eg: QTVREnableHotspotID 3

QTVRShowHotSpots show

Tells the QuickTime VR controller to show/hide all hots spots...

eg: QTVRShowHotSpots true

QTVRTranslateObject xMove yMove

This can be used to move around an object movie, once it is zoomed in a little.

eg: QTVRTranslateObject 5 5

QTVRSetViewState long viewStateType, short state

Sets the object node's state type to the new state value.

Misc ACTIONS

MusicPlayNote sampleDesc partNumber delay pitch velocity duration

eg: MusicPlayNote 1 2 0 60 100 600 TrackIndex=5

For this call to work, it needs to target a 'blank' midi track. That is, a midi track without any note information, but just instruments. If you just say MusicPlayNote ????? and do a script to movie, we will prompt you with a dialog that will add this track automatically and target it...

Don't confuse this kind of midi with just adding a standard midi track using the simple sound tool, this is for doing specific things when a hotspot is clicked. Play a sequence of notes, or a sound effect...

This script plays a major scale:

On MouseDown Hotspot 9

```
#####MusicPlayNote 1 2 0 60 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555 62 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*2 64 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*3 65 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*4 67 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*5 69 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*6 71 100 555 TrackIndex=5
#####MusicPlayNote 1 2 555*7 72 100 555 TrackIndex=5
```

end

□

"Supported Flags: none

Param1: [long sampleDescIndex]

Param2: [long partNumber]

Param3: [long delay]

Param4: [long pitch]

Param5: [long velocity]

Param6: [long duration]

This causes the target music track to play a note. Since the current selection of instruments for a music track is determined by its sample description, you use the sampleDescIndex parameter to select which sample description to use. The partNumber parameter specifies which part to use within the sample descriptions list. The default is 1.

If you want the note to be delayed, you may pass a positive value for the delay parameter, which is interpreted in the time scale of the music track. Pass 0 for no delay. The pitch parameter selects which note to play; for example, Middle C is 60, Middle B is 59. The velocity specifies the volume that the note is played at. The duration specifies the length of time that the note is played for and is interpreted in the time scale of the music track.

MusicSetController

"Supported Flags: none
Param1: [long sampleDescIndex]
Param2: [long partNumber]
Param3: [long delay]
Param4: [long controller]
Param5: [long value]
□

"Sets a controller value for a part in the target music track. Since the current selection of instruments for a music track is determined by its sample description, you use the sampleDescIndex parameter to select which sample description to use. The partNumber parameter specifies which part to use within the sample descriptions list.

If you want the controller change to be delayed, you may pass a positive value for the delay parameter, which is interpreted in the time scale of the music track. Pass 0 for no delay. Controller values control things, such as pitch bend and reverb. For more information about available controllers, see QuickTime Music Architecture.

Constants you can use for the 'controller' parameter:

volume 7
balance 8
pan 10
pitchBend 32"

```
kControllerModulationWheel = 1,  
kControllerBreath          = 2,  
kControllerFoot           = 4,  
kControllerPortamentoTime = 5, /* time in 8.8 seconds, portamento on/off is omitted,  
0 time = 'off' */  
kControllerVolume         = 7, /* main volume control */  
kControllerBalance        = 8,  
kControllerPan            = 10, /* 0 - "default", 1 - n: positioned in output 1-n (incl fractions)  
*/  
kControllerExpression     = 11, /* secondary volume control */  
kControllerLever1         = 16, /* general purpose controllers */  
kControllerLever2         = 17, /* general purpose controllers */  
kControllerLever3         = 18, /* general purpose controllers */
```

```

kControllerLever4      = 19, /* general purpose controllers */
kControllerLever5      = 80, /* general purpose controllers */
kControllerLever6      = 81, /* general purpose controllers */
kControllerLever7      = 82, /* general purpose controllers */
kControllerLever8      = 83, /* general purpose controllers */
kControllerPitchBend    = 32, /* positive & negative semitones, with 8 bits fraction,
same units as transpose controllers*/
kControllerAfterTouch   = 33, /* aka channel pressure */
kControllerPartTranspose = 40, /* identical to pitchbend, for overall part xpose */
kControllerTuneTranspose = 41, /* another pitchbend, for "song global" pitch offset */
kControllerPartVolume   = 42, /* another volume control, passed right down from note
allocator part volume */
kControllerTuneVolume   = 43, /* another volume control, used for "song global"
volume - since we share one synthesizer across multiple tuneplayers*/
kControllerSustain      = 64, /* boolean - positive for on, 0 or negative off */
kControllerPortamento  = 65, /* boolean*/
kControllerSostenuto    = 66, /* boolean */
kControllerSoftPedal    = 67, /* boolean */
kControllerReverb       = 91,
kControllerTremolo      = 92,
kControllerChorus       = 93,
kControllerCeleste      = 94,
kControllerPhaser       = 95,
kControllerEditPart     = 113, /* last 16 controllers 113-128 and above are global
controllers which respond on part zero */
kControllerMasterTune   = 114,
kControllerMasterTranspose = 114, /* preferred*/
kControllerMasterVolume = 115,
kControllerMasterCPULoad = 116,
kControllerMasterPolyphony = 117,
kControllerMasterFeatures = 118

```

If (expression) actionList endif

eg:

```

##### If theMovieTime>1000
#####SetPanAngle 5 delta
#####endif

```

While (expression) actionList endif

eg:

```

#####While theMovieTime>1000
#####SetPanAngle 5 delta
#####endif

```

GoToURL

eg: GoToURL "http:\\www.apple.com"

□

"Supported Flags:none

Param1: [C string]

If the movie is currently being presented using the Web browser plug-in, this causes the browser to go to the specified URL. If the movie is being presented by MediaPlayer, a Web browser is launched and goes to the specified URL. You may optionally specify a particular frame within a Web page.

To specify a URL, use the standard Web address formató for example, http://www.apple.com. You may optionally use angle bracketsófor example, <http://www.apple.com>. To specify a particular frame within a URL, you must use angle brackets followed by T<frameName>ófor example, <http://www.apple.com>T<frameName>."

□

"To target the QuickTime Player use:

<http://www.apple.com/some_movie.mov>T<QuickTimePlayer>

From QuickTime Player you can replace yourself with:

<http://www.apple.com>T<myself>

You can also specify the following target names, with the standard browser meanings:
_blank; _self; _top; _parent;"

// Open in QTPlayer... from browser.

GoToURL "<http://www.apple.com/npr1.mov>T<QuickTimePlayer>"

□

// Open in frame...

GoToURL ""<http://www.apple.com/npr1.mov>T<MovieFrame>"

□

// This opens the URL in the webbrowser

GotoURL ""

"http://www.pathfinder.com/time/reports/space/quicktime2/oldmissiontonewmission.mov"

□

// Open in QTPlayer

GotoURL ""<http://my.nice.server/test.mov>T<QuickTimePlayer>"

□

// Open in new browser window.

GotoURL ""<http://my.nice.server/test.mov>T<_blank>"

□

// Replace current QTPlayer window with new movie

```
// Will also replace current "embed" movie in browser with new one.  
GoToURL□□□'<A.mov>T<myself>"
```

SendQTEventToSprite

□

eg: SendQTEventToSprite {0,200,(0,0),0}□TrackIndex=5 SpriteIndex=2

"Supported Flags:none

Param1: [(SpriteTargetAtoms)]

Param2: [QTEventRecord theEvent]

Sends theEvent to the sprite specified by SpriteTargetAtoms. You may send any event to a sprite, including a custom event that you define. Note that kActionSendQTEventToSprite has no target, since it is handled by the system, although you do specify a target Param1 for the event that it sends Param2.

When sending a custom event, make sure the events constant does not conflict with an existing event. If you need to share a complicated event handler among many sprites, you may wish to define a custom event handler on a single sprite and have the others send it the custom event using kActionSendQTEventToSprite."

DebugStr

eg: DebugStr "WRONG_LOOP"

"Passes DebugString through the movie controller's filter proc using the mcActionShowMessageString constant.

Applications that wish to display the message string can use the movie controller's MCSetActionFilter routine along with the constant mcActionShowMessageString in order to receive the string."

QT Player can show you DebugStr variable[1,Track="SpriteTrack"] where the variable's value is shown.

"you should be aware that with the advent of QT 5, Eric Carlson was nice enough to support pop-up alerts when Debugstr is encountered (from the QT Plug-in).

You just have to go to Quicktime Preferences, choose media keys, then make a new key-value pair:

Category: QTPIShowDebugMessages

Key: alert"

PushCurrentTime

"Places the current movie time onto the top of the movie controller's stack of times. The movie controller maintains a stack data structure of movie times. The stack is manipulated using these actions."

□

PushCurrentTimeWithLabel

eg: PushCurrentTimeWithLabel "myTime"

□

Pushes the current time onto the top of the movie controller's stack of times, along with a label. This label is not a chapter name, it is a tag that can be used in conjunction with kActionPopAndGotoLabeledTime .

PopAndGotoTopTime

"Retrieves the top time from the movie controller's stack of times and removes it from the stack. The movie's current time is then set to the time retrieved. "

□

PopAndGotoLabeledTime

PopAndGotoLabeledTime "point1"

□

"Searches from the top of the movie controller's stack of times towards the bottom until it finds a time with the specified label. If found, it removes all times from the top of the stack through the labeled time and sets the movie's current time to this time. Note that only the topmost time with a matching label is popped, not all occurrences times with the same label. "

StatusString

eg: StatusString "http://www.apple.com" 1

You may use this action to instruct the QuickTime plug-in to display a URL link or other information in the status area of the browser. In order to do this, you set the flags parameter to kStatusStringIsURLLink and the statusString to a C string containing the URL or other information to display.

QuickTime Streaming uses the status string action to report some types of streaming status. Your application may listen to this status by installing a movie controller filter procedure. In this case, the kStatusStringIsStreamingStatus bit will be set in the flags field. Additionally, if the kStatusHasErrorCode bit is set, then the high 16 bits of stringTypeFlags is an error code number.

SendQTEventToTrackObject

□

SendQTEventToTrackObject {0,200,(0,0),0} TrackIndex=5

□

"Supported Flags: none

Parm1: [[[TrackObjectTargetAtoms]] trackTarget]

Parm2: [QTEventRecord theEvent]

Similar to kActionSendQTEventToSprite, but applies to any wired media handler that contains track objects such as Sprite, Text, and QuickDraw 3D. This lets you send standard or custom events to track objects."

AddChannelSubscription

eg: AddChannelSubscription "vrhotwiresTV" "http://www.vrhotwires.com" "picture"

"Supported Flags: none

Param1: [CString titleString]

Param2: [CString URL]

Param3: [CString pictureURL] (optional)

Adds a channel guide subscription of kBookmarkSubscriptionType with the supplied title, URL, and, optionally, a picture URL to the QuickTime Preferences file.

"

RemoveChannelSubscription

eg: RemoveChannelSubscription "vrhotwiresTV"

OpenCustomActionHandler

eg: OpenCustomActionHandler 1 {0,0,0,0,0}

□

"Supported Flags: none

Param1: [long handlerID]

Param2: [ComponentDescription desc]

This action attempts to locate and open a custom action-handling component with the supplied component description and assign it the specified handlerID. If successful, the handler is kept open so that its state is retained between calls. It will be closed automatically when the movie is disposed.

This call may fail if there is no component of the specified type available, or if there is already a custom handler open with the same handlerID. You may use the theUniqueCustomActionHandlerID to obtain a unique id for a handler you wish to open. You may use the theCustomActionHandlerIDsOpen to determine if a handler with a certain handlerID is open."

DoScript

eg: DoScript 1 "This_is_a_script?" "argument"

"kActionDoScript (long flags, CStr commands, CStr arguments)

This new wired action has no target (system target).

The action calls the root movie controller's mcActionDoScript, so that scripts can be invoked by a host application. For example, the QuickTime Plug-in in a browser can invoke JavaScripts.

If the script flags are set to kScriptIsUnknownType or kScriptIsJavaScript, the QuickTime Plug-in invokes a JavaScript routine in the HTML file that has embedded the QuickTime movie, with the following prototype:

```
function DoFSCommand(command, arguments) { }
```

If the movie is embedded with a NAME tag, a movieName tag, or is named by user data, then this prototype is used instead:

```
function movieName_DoFSCommand(command, arguments) { }
```

This allows for wired movies to invoke a JavaScript. The QuickTime Plug-in now also supports many movie-related JavaScript routines, so it is possible to set sprite track variables and post custom wired events to a wired movie from JavaScript as well. It is important to note that this functionality works only with the QuickTime Plug-in and that some versions of some browsers do not support the necessary interfaces to allow for these things to work. See the section "JavaScript Support" in this document for more details.

"mcActionDoScript allows a movie to send requests to execute scripts of various sorts to a host application. The parameter is a QTDoScriptPtr.

```
struct QTDoScriptRecord {  
long scriptTypeFlags;  
char *command;  
char *arguments;  
}; typedef QTDoScriptRecord *QTDoScriptPtr;
```

The scriptTypeFlags currently defined are:

```
kScriptIsUnknownType=0  
kScriptIsJavaScript=1,  
kScriptIsLingoEvent=2,  
kScriptIsVBEvent=3,  
kScriptIsProjectorCommand=4  
}; "  
□  
□
```

```
"// The <HTML> for this is here (remove "//" comments):  
// <SCRIPT LANGUAGE="JavaScript">  
// <!-- Allow old browsers to miss the JavaScript code;  
// // This gets called by the embeded movie with <embed moviename="TEST"...>  
// function TEST_DoFSCommand(command, arg1, arg2, arg3)  
// {  
// if (command == "Add") {  
// var arg_total = 0;  
// for (var i = 1; i < arguments.length; i++) {  
// arg_total = arg_total + arguments[i];  
// }  
// alert("Movie sum is: " + arg_total);  
// }  
// if (command == "Subtract") {  
// var arg_total = arguments[1];  
// for (var i = 2; i < arguments.length; i++) {  
// arg_total = arg_total - arguments[i];  
// }  
// alert("Movie difference is: " + arg_total);  
// }  
// }  
// // end of old browser avoidance -->  
// </SCRIPT>  
□
```

□ The vrhotwires Scripting code to call this, is for example:
□

```
// Results in 1+2+3 = 6  
DoScript 1 "Add" "1,2,3"
```

```
/ Results in 1-2-3 = -4  
DoScript 1 "Subtract" "1,2,3"  
□  
□
```

DoCompressedActions (compressed QTAtomContainer prefixed with eight bytes:
long compressorType, long decompressedSize)

□ _____

SendMessage (long appId)

eg:SendMessage 4

This allows you to send a wired movie message to a host application like
QuickTime Player.

The application message can be one of the following defines:

```
kQTAppMessageSoftwareChanged = 1 /* notification to  
app that installed QuickTime software has been updated*/
```

```
kQTAppMessageWindowCloseRequested = 3 /* request for app  
to close window containing movie controller*/
```

```
kQTAppMessageExitFullScreenRequested = 4/* request for app  
to turn off full screen mode if active*/
```

□ _____

LoadComponent (ComponentDescription handlerDesc)

Allows a script to make sure that a component is available by using this call,
and then later calling `theComponentVersion` to verify that it loaded. This
helps with managing component downloads.



SetFocus [(TargetAtoms theObject)]

Sets the current focus. QuickTime supports Flash text edit characters, text tracks, and sprites. The sprite objects and text tracks may be targeted in the normal way. Flash edit characters can be targeted in a similar way to sprites, but only the text character object ID is supported with this release. The following defines are available:

```
enum {
    kTargetObjectName = FOUR_CHAR_CODE('obna'), /*
(PString objectName) */
    kTargetObjectID   = FOUR_CHAR_CODE('obid'), /*
(QTAtomID objectID) */
    kTargetObjectIndex = FOUR_CHAR_CODE('obin') /* (short
objectIndex) */
};
```



DontPassKeyEvent no params

```
kActionDontPassKeyEvent = 6163, /* no params */
```

This is for key event handlers to notify the controller that they have handled the event -- don't pass up the chain.

The best example of usage of this wired action is in a sprite `keyEvent` handler. When the sprite is using the arrow keys for movement, you can use this action to tell the controller not to pass the arrow keys up the chain. Otherwise, the controller may use the keys to change the time, sound level, and so on.



SetRandomSeed randomSeed

Sets the QuickDraw seed value which is starting point for any subsequent `theRandom` calls.

SpriteTrack Actions

SpriteTrackSetVariable variable value

eg: **SpriteTrackSetVariable 123 4567**

□

In vrh there are two places you can store and retrieve your variables.
In a sprite tracks, or in a flash track.

Since sprites have been around since qt3 and flash variables just came about in qt5,
it's much more common to use sprite tracks...

It's easy to create a sprite track in vrh, just drag an image onto the movie and you will be
asked if it should be a sprite or a video track...

Make sure from the 'time' panel in movie info that your sprite track is the same duration as
your movie, especially if its a multinode and you want to retrieve values from the sprite track
in another node...

A synonym for SpriteTrackSetVariable is just SetVariable... and using numbers to identify
the variable is what qt supports.

vrh will allow you to use names for your variables if you define them in the frameloaded
event...like

```
//define (1234)=(joe)
```

now you can say: SetVariable joe 42 Tracktype=sprite
and to get it out you could say: if variable[joe]=1

By looking at the scripts for some examples you should find this fairly easy.

"sprite variables

are runtime only, they are not saved and restored. If you need to initialize variables to
certain values, you can do it during the FrameLoaded event."

SpriteTrackNewSprite spriteID imageIndex matrix visible layer graphicsMode

eg:SpriteTrackNewSprite 7 6 1 0 0 0 1 0 600 0 16384 1 neg5 {0,{0,0,0}} 7

"Supported flags: none

Param1: [QTAtomID spriteID]

Param2: [short imageIndex]

Param3: [MatrixRecord matrix]

Param4: [short visible]
Param5: [short layer]
Param6: [ModifierTrackGraphicsModeRecord
graphicsMode]
Param7: [QTAtomID actionHandlingSpriteID]
Dynamically creates a new sprite with the properties
specified by the parameters. You may pass zero for the
spriteID and a unique ID will be assigned."

"A sprite created at runtime is associated with the current sprite key frame. It will remain alive until a new sprite key frame is loaded. It has all the regular properties of a sprite including the new sprite property `kSpritePropertyActionHandlingSpriteID` which delegates its handling of wired events to another sprite. In QuickTime 4, runtime sprites cannot have their own unique event handling atoms."

SpriteTrackDisposeSprite 1

"Disposes the sprite with the specified ID. Note that this disposes the sprite object in memory, it does not dispose the sprite from the media. If you dispose a sprite that exists in a sprite track's media, it will be created again the next time the key frame sample is loaded."

SpriteTrackSetVariableToString

eg: `SpriteTrackSetVariableToString 1004 "Turtle" TrackIndex=5`

"Supported flags: none Param1: [QTAtomID variableID] Param2: [CString theString] Sets the value of the sprite track variable specified by `variableID` to `theString`. This replaces the previous value of the variable. Variables may be either a string or a floating-point number."

SpriteTrackConcatVariables

eg: `SpriteTrackConcatVariables 123 124 125 TrackIndex=5`

"Supported flags: none
Param1: [QTAtomID firstVariableID]
Param2: [QTAtomID secondVariableID]
Param3: [QTAtomID resultVariableID]
Concatenates the string in the sprite track variable specified

by secondVariableID to the end of the string in firstVariableID and places the result in the variable resultVariableID. Uninitialized string variables are the empty string. Variables containing floating-point numbers are coerced to strings."

SpriteTrackSetVariableToMovieURL

eg:SpriteTrackSetVariableToMovieURL 123
Movie="http://www.vrhotwires.com/cool.mov"
□

"Sets the value of a sprite track variable to a string containing the URL of the movie that contains the sprite track, or optionally the URL of an external movie if the second parameter is used.

For example if the URL of the movie is:

http://www.apple.com/media/quicktime/excitement.mov

this command will set a variable to:

http://www.apple.com/media/quicktime/excitement.mov"

SpriteTrackSetVariableToMovieBaseURL

eg:SpriteTrackSetVariableToMovieBaseURL 123
Movie="http://www.vrhotwires.com/cool.mov"
□

"Sets the value of a sprite track variable to a string containing the Base URL of the movie that contains the sprite track, or optionally the URL of an external movie if that target is supplied.

For example if the URL of the movie is:

http://www.apple.com/media/quicktime/excitement.mov

this command will set a variable to:

http://www.apple.com/media/quicktime"

SpriteTrackSetAllSpritesHitTestingMode

no params:

to quote apples docs:

"The following new qt6 actions enable interactive content creators to request, display, and manage images hosted outside of the Movie in which they are used.

The two new actions, `kActionSpriteTrackNewImage` and `kActionSpriteTrackDisposeImage`, always interact with the images loaded at runtime, and should always be used to reference indexes higher than those of the images that are integrated within the movie when it is created.

`kActionSpriteTrackNewImage` takes as a parameter the URL of the image to be requested and an ID with which you can reference that image. Passing an ID of 0 will prompt this action to assign the next available (unique) ID greater than the current image count. In comparison, the index assigned will always be the integer one greater than the current image count.

For example, a target sprite track has 2 images with index/ID pairs of 1/1 and 2/777, respectively, before this action is executed. The new image action is called with "image.jpg" as the URL and a `desiredID` of 6. Assuming the URL is valid, the new image action will be given the index 3 and honor the requested ID of 6. If the URL is invalid, or if ID 1 or 777 is requested, index 3 will not be assigned nor will any ID, because the image has failed to load. A subsequent call of `kActionSpriteTrackNewImage` will attempt to use index 3 again. For this reason, it is advantageous for the movie author to maintain a tally or query the number of images in the Track (`theSpriteTrackNumImages`) to predict the new index.

`kActionSpriteTrackDisposeImage` takes as a parameter the index of the image to be released from memory. The image specified by the index is required to be one loaded through `kActionSpriteTrackNewImage`. In other words, the index is required to be one previously assigned by a `kActionSpriteTrackNewImage`. Images authored into the movie, either as data or by reference, cannot be disposed of in this way. Note also that subsequent calls to `kActionSpriteTrackNewImage` will not fill the "holes" left by `kActionSpriteTrackDisposeImage`, but will continue to increment the index. Thus, `kActionSpriteTrackDisposeImage` exists to enable movie authors to manage the memory usage of a movie during playback—for example, when the movie may only need an externally referenced image temporarily.

SpriteTrackNewImage imageURL, desiredID

Loads an image and gives it the next available image index, and the desired ID, if available.

see paragraph above

SpriteTrackDisposeImage imageIndex

see paragraph above

ApplicationNumberAndString

none Param1: [long aNumber] Param2: [Str255 aString]

QuickTime does nothing when this action is executed; it is intended to be used by applications that wish to "hook" it, using the movie controller's `MCSetActionFilter` routine with the new constant `mcActionExecuteOneActionForQTEvent`. Applications may look at the number and string parameters to determine what they want to do when the action is

QD3DNamedObjectTranslateTo

eg:QD3DNamedObjectTranslateTo 5 5 5 TrackIndex=1 Object="Joe

Supported flags: none Param1: [Fixed x] Param2: [Fixed y] Param3: [Fixed z] Translates the named object to (x, y, z).

QD3DNamedObjectScaleTo

eg:QD3DNamedObjectScaleTo 5 5 5 TrackIndex=1 Object="Joe

Supported flags: none Param1: [Fixed xScale] Param2: [Fixed yScale] Param3: [Fixed zScale] Scales the named object by (xScale , yScale , zScale).

QD3DNamedObjectRotateTo

eg:QD3DNamedObjectRotateTo 5 5 5 TrackIndex=1 Object="Joe

Supported flags: none Param1: [Fixed xDegrees] Param2: [Fixed yDegrees] Param3: [Fixed zDegrees] Rotates the named object to (xDegrees , yDegrees , zDegrees).

FLASH ACTIONS

FlashTrackSetPan x y targetTrack

eg: FlashTrackSetPan 123 456 TrackIndex=2

"

Supported flags: none

Param1: [short xPercent]

Param2: [short yPercent]

Pans a Flash Track by the specified x and y percentages."

FlashTrackSetZoom

□

eg:FlashTrackSetZoom 123 TrackIndex=2

□

□

"Supported flags: none Param1: [short zoomFactor]

Zooms a Flash track by zoomFactor ."

FlashTrackSetZoomRect

□

eg: FlashTrackSetZoomRect 100 100 200 200 TrackIndex=5

□

Because flash is largely a vector format, scaling an image is quite useful...

"Supported flags: none Param1: [long left] Param2: [long top] Param3: [long right] Param4: [long bottom]

Zooms a Flash track to the specified rectangle.

FlashTrackGotoFrameNumber

eg: FlashTrackGotoFrameNumber 123 TrackIndex=3

Supported flags: none Param1: [long frameNumber] Sets the movie time of the movie containing the Flash track to the time that corresponds to the specified flash frame number.
□

FlashTrackGotoFrameLabel label

eg: FlashTrackGotoFrameLabel "Yippee" TrackIndex=5

"Supported flags: none Param1: [CString frameLabel] Sets the movie time of the movie containing the Flash track to the time that corresponds to the flash frame with label equal to frameLabel .

FlashTrackSetFlashVariable path name value updateFocus

eg: FlashTrackSetFlashVariable "" "buttonPressed" "32" false TrackIndex=1

Sets a Flash variable to the value.

FlashTrackDoButtonActions path buttonID transition

eg: FlashTrackDoButtonActions "" 9 3 TrackType=Flash

Sends a message to a Flash button to perform a transition. This causes whatever Flash or QuickTime action associated with the button to perform.

□

MovieTrackAddChildMovie

eg:MovieTrackAddChildMovie 12 "www.apple.com/coolmovie.mov"

□

"This action adds a new movie URL data reference to the targeted movie track's array of movie data references.
The URL data reference is added with the specified ID. If a data reference with the same ID already exists, it is replaced.

It is generally a good idea to use an ID that is not contained in the movie track's sample, since this may be reloaded under some conditions."

□'The movie media handler maintains a dynamic array of movie data references. Each data reference is assigned a unique ID. These data references may be used to load a new movie into the movie track. A movie media sample can initialize this array with any type of data references."

□

MovieTrackLoadChildMovie movieID

eg:MovieTrackLoadChildMovie 12

□

"This action loads a movie specified by childMovieID as the current movie being played by the movie track. The movie replaces the current movie.

Another new action operates on a root movie or a movie track target:"

□

□

MovieTrackLoadChildMovieWithQTLISTParams QTAtomID
childMovieID, C string qtlisXML)

□

TEXT TRACK ACTIONS

TextTrackPasteText theText startSelection endSelection

eg: TextTrackPasteText "hello" 1 300 TrackIndex=4

Replaces a selection in the text track with theText.

□

TextTrackSetTextBox (short left, short top, short right,
short bottom)

Changes the textBox of text track to the passed in size.

□ _____

TextTrackSetTextStyle textStyle
Changes text track style - a TextStyle record.

□ _____

TextTrackSetSelection startSelection endSelection
Sets the text track selection.

□ _____

TextTrackSetBackgroundColor backgroundColor
Sets the text track background color.

□ _____

TextTrackSetForegroundColor foregroundColor

Sets the text color.

□ _____

TextTrackSetFace fontFace

Sets the text face (style) of all text.

□ _____

TextTrackSetFont fontID
Sets the text font of all text.

□ _____

TextTrackSetSize fontSize

Sets the text size of all text.

□ _____

TextTrackSetAlignment alignment

Sets the text alignment as in:

teJustLeft = 0,

teJustCenter = 1,

```
teJustRight = -1,  
teForceLeft = -2, /* new names for the  
Justification (word alignment) styles */  
teFlushDefault = 0, /*flush according to the line  
direction */  
teCenter = 1, /*center justify (word alignment) */  
teFlushRight = -1,/*flush right for all scripts */
```

□

TextTrackSetHilite startHighlight endHighlight highlightColor
Highlights from the startHighlight offset to the endHighlight offset.

□

TextTrackSetDropShadow dropShadow transparency
Sets the drop shadow parameters. This only works if displayFlags
has been set
with dfDropShadow.

□

TextTrackSetDisplayFlags flags

Sets the text display flags as in:

```
dfDontDisplay = 1 << 0,/* Don't display the text*/  
dfDontAutoScale = 1 << 1,/* Don't scale text as track bounds or  
shrinks*/  
dfClipToTextBox = 1 << 2,/* Clip update to the textbox*/  
dfUseMovieBGColor = 1 << 3,/* Set text background to movie's  
background color*/  
dfShrinkTextBoxToFit = 1 << 4,/* Compute minimum box to fit the  
dfScrollIn = 1 << 5,/* Scroll text in until last of view */  
dfScrollOut = 1 << 6,/* Scroll text out until last of gone (if both  
set, scroll in then out)*/  
dfHorizScroll = 1 << 7,/* Scroll text horizontally (otherwise it's  
vertical)*/  
dfReverseScroll = 1 << 8,/* vert: scroll down rather than up;  
horiz:  
scroll backwards (justfication dependent)*/  
dfContinuousScroll = 1 << 9,* new samples cause previous
```

```
samples scroll out */
dfFlowHoriz = 1 << 10,/* horiz scroll text flows in textbox
rather than extend to right */
dfContinuousKaraoke = 1 << 11,/* ignore begin offset, hilite
everything
up to the end offset(karaoke)*/
dfDropShadow = 1 << 12,/* display text with a drop shadow */
dfAntiAlias = 1 << 13,/* attempt to display text anti aliased*/
dfKeyedText = 1 << 14,/* key the text over background*/
dfInverseHilite = 1 << 15,/* Use inverse hiliting rather than using
hilite color*/
dfTextColorHilite = 1 << 16 /* changes text color in place of
hiliting.
```

□

TextTrackSetScroll (long delay)

Sets the time delay for start of the scroll. This only works when scroll flags are set in displayFlags.

□

TextTrackRelativeScroll (short deltaX, short deltaY)

Scrolls the text in the text box by the delta amounts.

□

TextTrackFindText (long flags, Str255 theText, ModifierTrackGraphicsModeRecord highlightColor)

Finds text in the track. Similar in operation to TextMediaFindNextText since this is what it uses.

□

TextTrackSetHyperTextFace (short index, long fontFace)

Sets the text face (style) of the indexed hypertext.

□

TextTrackSetHyperTextColor (short index, ModifierTrackGraphicsModeRecord highlightColor) */

Sets the text color of the indexed hypertext.



TextTrackKeyEntry character
Replaces the selection with the character.



TextTrackSetEditable editState

Controls the key entry state of the text track:

```
#define kKeyEntryDisabled 0
```

```
#define kKeyEntryDirect 1
```

```
#define kKeyEntryScript 2
```

kKeyEntryDisabled is default.

If kKeyEntryDirect is on, then key events are passed directly to the text track.

If kKeyEntryScript is on, then scripted mouse and key events are allowed.



TextTrackMouseDown

Passes the mouse click to the text track, which allows for selecting text or an insertion point when kKeyEntryScript is turned on.

QTLists

QTLists are hierarchical data structures stored in movies or tracks. Any movie or track can have a qtlist. QTLists resemble XML, with elements that can have child elements or values, as well as attributes. A set of wired actions and operands provide access to manipulate these lists and exchange them with servers via XML.

ListSetFromURL (C string url, C string targetParentPath)
This allows the scripter to use an XML file to initialize a list. Note that this is asynchronous action while ListServerQuery is an asynchronous method of loading lists.

ListAddElement (C string parentPath, short atIndex, C string newElementName)
Adds the element to the target list.

ListRemoveElements (C string parentPath, short startIndex, short endIndex)
Removes the element from the target list.

ListSetElementValue (C string elementPath, C string valueString)
Sets the list element value.

ListPasteFromXML (C string xml, C string targetParentPath, short startIndex)
Pastes an XML-formatted list into the target list at startIndex.

ListSetMatchingFromXML (C string xml, C string targetParentPath)
Replaces the matching element values in the target list.

ListServerQuery (C string url, C string keyValuePairs, long flags, C

string parentPath)

This provides a versatile method for sending and receiving data from a server. Note that this is an asynchronous method of loading data, and that the returned data will be available in the local “event.list” of the ListReceived event.

If the keyValuePairs string is non nil, it is appended first after the URL. The following flags are then in play:

```
enum {  
kListQuerySendListAsXML = 1,  
kListQuerySendListAsKeyValuePairs = 2,  
kListQueryWantCallback = 4,  
kListQueryDebugTrace = 8  
};
```

If kListQuerySendListAsXML or kListQuerySendListAsKeyValuePairs are on, then

the list target is used in the manner selected, and appended to the URL. The key

value pairs are appended in the following way:

url?key1=one&key2=two&key3=three.

XML is appended as follows:

```
url?qtlist=<qtlist><key1>one</key1><key2>two</key2><key3>t  
hree</key3></  
qtlist>
```

or

```
url?user=joe&qtlist=<qtlist><key1>one</key1><key2>two</key  
2><key3>three</  
key3></qtlist>
```

kListQueryWantCallback indicates whether a list received event is wanted to

receive data from the server.

kListQueryDebugTrace is added for authoring. This triggers a kActionDebugStr,

so that the application can see what URL was actually sent.

As with ListExchangeLists, the URL is encoded before delivery.

OPERANDS- What's an operand?

It's a reserved value you can get from quicktime. That is, a little number kept in a cubby hole somewhere, that knows something, like the time of day, or the current pan angle.

If we are used to saying `QTVRSetPanAngle 99`
we could set the Pan to the tilt by saying: `QTVRSetPanAngle theTiltAngle`
□

In vrhotwires we use 'the' to designate an operand. Other scripting systems might use 'Get' like `GetTiltAngle()`

In vrh 1.1 we accept operands with 'Get' as good input. So you can say `theMovieTime` or you can say `GetMovieTime` and they will both do the job. On writing out, it's always `theMovietime` though...
□

ALL OPERANDS

theSubscribedToChannel channelsURL

Returns `true` if the QuickTime Preference file contains a subscription to the channel specified by `channelURL`; otherwise returns `false`.

theUniqueCustomActionHandlerID

Returns an unused custom action handler ID. This value may be used with `OpenCustomActionHandler` and stored in a sprite track variable.

theCustomActionHandlerIDsOpen ID

Returns `true` if a custom action handler with the specified ID is currently open.

theConnectionSpeed

Returns the connection speed setting in a users QuickTime preferences file.

theGMTDay

Returns the day value (1..31) for the current Greenwich Mean time.

theGMTMonth

Returns the month value (1..12) for the current Greenwich Mean Time time

theGMTYear

Returns the year value for the current Greenwich Mean time.

theGMTHours

Returns the hours value in 24 hour time (0..23) for the current Greenwich Mean time.

theGMTMinutes

Returns the minutes value (0..59) for the current Greenwich Mean time.

theGMTSeconds

Returns the seconds value (0..59) for the current Greenwich Mean time.

theLocalDay

Returns the day value (1..31) for the current local time.

theLocalMonth

Returns the month value (1..12) for the current local time.

theLocalYear

Returns the year value for the current local time.

theLocalHours

Returns the hours value in 24 hour time (0..23) for the current local time.

theLocalMinutes

Returns the minutes value (0..59) for the current local time.

theLocalSeconds

Returns the seconds value (0..59) for the current local time.

theRegisteredForQuickTimePro

Returns true if the user is registered for the Pro version of QuickTime, otherwise returns false.

thePlatformRunningOn

Returns 1 if the computer platform is Macintosh, 2 if it is Windows.

theQuickTimeVersion This allows the scripter to verify that the executing QuickTime has been registered.

theComponentVersion type,subType, manufacturer
Returns the version of a specific QuickTime component. The component is specified using four character string codes for the type, subType, and manufacturer. If the component is not available, a version number of zero is returned.

theOriginalHandlerRefcon
Returns the refcon of the original event handler. This is useful if you are using multiple sprites that delegate the handling of actions to a common handler using the new `actionHandlerID` property. The original handler's refcon in this case is the ID of the sprite that was originally clicked on or sent a custom event.

theTicks
This is the system ticks. It is updated 60 times a second.

theMaxLoadedTimeInMovie
This is the useful operand to tell how much of a movie has loaded. It seems to update every sample, or sometimes once a second...

theEventParameter index

Allows key and mouse event handlers to get parameters of the triggered event.

For the mouse:

- 1 : where.h
- 2 : where.v
- 3 : modifiers

For the key:

- 1 : where.h
- 2 : where.v
- 3 : modifiers
- 4 : key
- 5 : scancode

theFreeMemory
Returns the amount of memory free in the application heap.

theNetworkStatus

theNetworkStatus

Returns the status code of the network connection:

```
kQTNetworkStatusNoNetwork = -2,  
kQTNetworkStatusUncertain = -1,  
kQTNetworkStatusNotConnected = 0,  
kQTNetworkStatusConnected = 1
```

theQuickTimeVersionRegistered version

Returns the version of QuickTime that is in use.

The format is major_version.minor_version. So, for example, QuickTime 4.0, 4.0.1, 4.0.2 and 4.0.3 all return 4.0

Note: this operand is only available in QuickTime 4.0 and above, so for now it's a bit dull." □

theSystemVersion

This operand returns the Macintosh version. On Mac OS 9.04, for example, this

would be hexadecimal 904, a 32-bit number converted to a float.

On Windows, it

currently returns 0. The high half has several flags in it: the bit that's set if

you're on Windows 9x and another bit that's set if you're on Windows NT. The

defines, which are in Movies.h, are as follows:

```
/* flags for theSystemVersion*/  
enum {  
kSystemIsWindows9x = 0x00010000,  
kSystemIsWindowsNT = 0x00020000  
};
```

theMovieVolume

The target movie's current volume level is returned.

theMovieRate

The target movie's current rate is returned.

theMovieIsLooping

If the target movie is in looping mode a value of 1 is returned; otherwise, a value of 0 is returned.

theMovieLoopsPalindrome

If the target movie is in palindrome mode, a looping value of 1 is returned; otherwise, a value of 0 is returned.

theMovieTime

The target movie's current time value is returned.

theMovieDuration Returns the duration of the target movie in movie time scale units.

theMovieTimeScale Returns the timescale of the target movie.

theMovieWidth Returns the current width of the target movie.

theMovieHeight Returns the current height of the target movie.

theMovieLoadState

Returns the load state of the target movie. <0 indicates an error.

kMovieLoadStateLoading = 1000,

kMovieLoadStatePlayable = 10000,

kMovieLoadStatePlaythroughOK = 20000,

kMovieLoadStateComplete = 100000L

theMovieTrackCount

Returns the track count of the target movie.

theMovieIsActive

theMovieName Gets the target movie name, if any, stored in the user data as type 'plug' with data "moviename=theActualMovieName" or the name the app has defined for the movie.

theMovieID

Gets the target movie name, if any, stored in the user data as type 'plug' with data "pmovieid=##".

theMovieChapterCount

Gets the chapter count.

theMovieChapterIndex

Gets the current chapter index.

theMovieChapterName

Gets the current chapter name.

theMovieChapterNameByIndex index

Gets a chapter name by index...so if your fourth chapter is 'intro' you can say theMovieChapterNameByIndex[4]

theMovieChapterIndexByName name

Gets the chapter index from the input name..
theMovieAnnotation requested flags
theMovieConnectionFlags
theMovieConnectionString

theTrackVolume The current volume level of the target track is returned.

theTrackBalance The current balance setting of the target track is returned.

theTrackEnabled
The target track's enabled state is returned. A value of 1 is returned if it is enabled, and 0 if it is not.

theTrackLayer The target track's layer is returned.

theTrackWidth Returns the current width of the target track.

theTrackHeight Returns the current height of the target track.

theTrackDuration Returns the duration of the target track.

theTrackName Gets the target track name stored in track user data as type 'name'.

theTrackID Gets the target track ID.

theTrackIdleFrequency Gets the target track's current idle frequency -- only sprite and text tracks are currently supported.

theTrackBass Gets the track's bass value. The target should be a sound track.

theTrackTreble Gets the track's treble value. The target should be a sound track.

theSpriteBoundsLeft
Gets the left border of the sprite

theSpriteBoundsTop
Gets the top border of the sprite

theSpriteBoundsRight
Gets the right border of the sprite

theSpriteBoundsBottom
Gets the bottom border of the sprite

theSpriteImageIndex
Gets the current image index of the sprite. So if sprite 1 is set to show the image of sprite 2 you can find out...

theSpriteVisible Gets the visibility state of the sprite...

theSpriteLayer Gets the current layer of the sprite

theSpriteTrackVariable variableID

Gets the value of a variable. In vrh this can be put in and read out as variable. For example: SetPanAngle variable[1]

theSpriteTrackNumSprites returns the number of sprites in this keyframe

theSpriteTrackNumImages returns the number of images in this keyframe...

theSpriteID returns this sprite's ID

theSpriteIndex returns this sprite's index.

These operands return the location of the four corners of the sprite:

theSpriteFirstCornerX

theSpriteFirstCornerY

theSpriteSecondCornerX

theSpriteSecondCornerY

theSpriteThirdCornerX

theSpriteThirdCornerY

theSpriteFourthCornerX

theSpriteFourthCornerY

theSpriteImageRegistrationPointX

theSpriteImageRegistrationPointY

theSpriteTrackSpriteIDAtPoint x y

Returns the ID of the sprite that would be hit at the point where a mouse click occurred in the target sprite.

theSpriteName Gets the target sprite name.

theSpriteCanBeHitTested short

theSpriteTrackAllSpritesHitTestingMode

theSpriteTrackImageIDByIndex imageIndex

theSpriteTrackImageIndexByID = 3108, /* QTAtomID */

theQTVRPanAngle

Returns the current pan angle of the qtvr movie. It is a number in degrees from 1 to 360

theQTVRTiltAngle

Returns the current tilt angle of the qtvr movie. Remember that when the movie is zoomed in, it will be able to tilt more...

theQTVRFieldOfView

Returns the current FOV angle of the current movie.

theQTVRNodeID

Returns the current Node ID of the movie. So in a multinod you can find out what node the viewer is in...

theQTVRHotSpotsVisible Returns whether the QuickTime VR controller is displaying the hot spots.

theQTVRViewCenterH

Returns the view centerH of an QuickTime VR object controller.

theQTVRViewCenterV

Returns the view centerV of an QuickTime VR object controller.

theQTVRViewStateCount

theQTVRViewState **viewStateType**

theMouseLocalHLoc **[TargetAtoms aTrack]**

Returns the current location of the mouse, on the horizontal plane.

theMouseLocalVLoc **[TargetAtoms aTrack]**

Returns the current location of the mouse, on the vertical plane.

theKeysDown **[short modKeys, char asciiValue]**

"This operand returns:

1 if the key with the specified lower case ASCII character code is currently pressed, and 0 if it is not pressed.

If you specify modifier keys, these keys must also be pressed for 1 (true) to be returned.

Note that if extra modifier keys are pressed, a positive indication will still

be returned. The logic is "are these keys currently pressed?", not "are only these keys pressed?". Additionally, you may determine if the mouse button is pressed.

You may pass 0 for modifier keys if you don't care about the modifier keys, or 0 for the ASCII character if you only care about the modifier keys. Passinf (0,0) always returns true.

The following flags may be used. Note the mapping of the flags for Windows machines.

Also, be warned that using the Control key may be a bad idea if you plan on using the movie on a Windows machine due to the fact that it is mapped to the Alt key, which is used by the Windows operating system.

btnState The mouse button is pressed. Windows: the left mouse button is pressed.

cmdKey The command key is pressed. Windows: the Ctrl key is pressed.

shiftKey The Shift key is pressed. Windows: The Shift key is pressed.

alphaLock The Caps Lock key is pressed. Windows: The Caps Lock key is pressed.

optionKey The option key is pressed. Windows: both the Ctrl and the Alt keys are pressed.

controlKey The control key is pressed. Windows: the Alt key is pressed.

Command = 256

Shift = 512

CapsLock = 1024

Option = 2048
Control = 4096
MouseButton = 128? "

theRandom [short min, short max]

Returns a random number... This number won't reset unless you use the RandomSeed action.

theCanHaveFocus [(TargetAtoms theObject)]

theHasFocus [(TargetAtoms theObject)] Keyboard Focus
Keyboard focus allows QuickTime 5 users to tab and click between
editable

fields and the movie controller. Editable fields are those that
support key events

-- for example, the edit text characters in Flash 4 content, text
and sprite tracks

(both of which are new with QuickTime 5).

To help QuickTime arbitrate focus when the Tab key is entered, a
new media

property has been defined:

```
enum {  
kTrackFocusCanEditFlag = 'kedt' // Boolean  
};
```

QuickTime will search for the next track that has this flag turned
on. QuickTime

will then query the track to see if it really wants focus. For
example, a Flash

track may not have any edit text characters, and therefore not
want the focus.

Once a track is found, it is assigned focus, and it determines how
this is to be shown to the user. If no track is found, then the focus
is returned to the movie controller.

QuickTime is making the following assumptions:

Flash media defaults to true in the absence of the above
property.

Text media defaults to false.

Sprite media defaults to false.

The major change from QuickTime 5 Public Preview releases is that

the Text
track needs to have the above property turned on as well as
having the
hasActions flag turned on for direct keyboard entry.
theCanHaveFocus = 5124, /* [(TargetAtoms theObject)] */
Does the track/object allow focus?
theHasFocus = 5125, /* [(TargetAtoms theObject)] */
Does the track/object have focus?

theTextTrackEditable

Returns the current key entry state of the target text track.

theTextTrackCopyText startSelection endSelection

Returns the selection range as a string of the target text track.

theTextTrackStartSelection

Returns the current starting selection point of the target text track.

theTextTrackEndSelection

Returns the current ending selection point of the target text track.

theTextTrackTextBoxLeft Returns the left edge of the text box of target text track in text track coordinates.

theTextTrackTextBoxTop Returns the top edge of the text box of target text track in text track coordinates.

theTextTrackTextBoxRight Returns the right edge of the text box of target text track in text track coordinates.

theTextTrackTextBoxBottom Returns the bottom edge of the text box of the target text track in text track coordinates.

theTextTrackTextLength

theListCountElements (parentPath) Returns the number of elements in the target list.

theListGetElementPathByIndex (parentPath, index)

Returns the name string of the element found a parentPath and the index of the target list.

theListGetElementValue (C string elementPath)

Returns the value of the element at elementPath of the target list.

theListCopyToXML (C string parentPath, long startIndex, long endIndex)

theListCopyToXML (C string parentPath, short startIndex, short endIndex)

Returns the selection of the target list as a XML-formatted string.

theSin /* float x */

theCos /* float x */

theTan = 8194, /* float x */

theATan /* float x */

theATan2 /* float y, float x */

theDegreesToRadians /* float x */

theRadiansToDegrees /* float x */

theSquareRoot /* float x */

theExponent /* float x */

theLog /* float x */

Math operands

theSin float x

theCos float x

theTan float x

theATan float x

theATan2 float y, float x

theDegreesToRadians float x

theRadiansToDegrees float x

theSquareRoot float x

theExponent float x

theLog float x

theFlashTrackVariable [CString path, CString name]

returns the value of the flash track variable as in
SetPanAngle theFlashTrackVariable["", "varname"]

theStringLength (C string text)

returns the length of a string

theStringCompare aText bText caseSensitive diacSensitive

Returns 0 for false, non-zero for true, indicating if the text is equivalent.

theStringSubString text offset length

Returns substring of text starting at 0-based offset 'offset' for length 'length'.

theStringConcat **aText bText**

□ Returns string produced by concatenating aText with bText.

□