Appendix I - QScript Reference

Introduction

Welcome to the LiveStage Professional QScript Reference Guide. This guide is designed to help you and provide more detailed information on the various QScript statements utilized by LiveStage Professional. It includes correct Syntax, Descriptions and Examples of each statement to assist you in understanding what they do and how to use them.

Conventions Used

`language element`	Text shown in this font must be entered exactly as shown.
`placeholder`	Text shown in italics indicates a place holder that you must replace with an appropriate value.
[optional]	Brackets indicate that the enclosed language elements are optional. However, brackets in bold are part of the syntax and must be entered as shown.
A | B | C	A vertical bar separates a group of elements from which one must be chosen.
<>	Enclose a group of multiple choice elements.

For example:

SetImageIndexTo(*index****)*** *[****min(****number****)*** ***max(****number****)*** ***Wraparound****]*

You can replace *index* with any numeric expression you want, such as

```
SetImageIndexTo(1)
```

Here are some examples:

```
// causes the image index to increase by one and when
// it passes 10 it starts over at 1
SetImageIndexBy(1) MIN(1) MAX(10) Wraparound
// causes the image index to be set to the value of x
// and limited to a number from 8 to 10
SetImageIndexTo(x) MIN(8) MAX(10)
```

QScript Syntax

This section of the QScript Reference contains descriptions of the various types of QScript syntactical constructs.

Data Specifiers

Data is the life blood of scripting languages. QScript is no exception. The following section describes the syntax for specifying data within QScript.

Constants	**Definitions**
Description	Constants, as the name implies, are data that can not be altered when the script is executed. Constants are also called "Literals". The following are some examples of constants in QScript:
Example	`1234` Numeric constant indicating the whole number 1234
	`1234.56` Numeric constant indicating the real number 1234.56
	`-1` Negative numeric constant indicating the whole number -1
	`"Hi"` Text or String constant containing the characters Hi
	`TRUE` Predefined boolean constant for TRUE
	`FALSE` Predefined boolean constant for FALSE
	`$"Image One"` The index of the image named "Image One"
	Constants can be used any place in a script where a number or string is required.
See also	Numeric Expression, String Literal

Numeric Expression — Definitions

Description

A Numeric Expression is an expression that, when evaluated, generates a numeric result. Numeric Expressions are made up of numeric elements (i.e. numeric constants, variables and properties) joined together with numeric operators (i.e. +, -, *, /, etc.).

Numeric Expressions can be used any place in a script where a numeric value is required.

Example

The following are examples of Numeric Expressions:

```
1 + 2 * 3
ThisSprite.ImageIndex + 1
ABS -1
```

See also

Constants, Operators, Properties, Variables

String Literal — Definitions

Description

A String Literal is made up of text characters enclosed within quotation marks (“). String Literals are used to specify textual information any place within scripts where a textual value is required.

Example

The following are examples of String Literals:

```
“Hello World!”
“http://www.totallyhip.com”
```

QT Version

4.0 or later

See also

Constants

Boolean Literal — Definitions

Description A Boolean Literal can be one of two values, **TRUE** or **FALSE**. Boolean Literals can be used any place in a script where a boolean value is required.

Example

```
SetVisible(TRUE)
```

See also Constants

Defines — Definitions

Description A define is an abstract representation of a constant. Defines allows you to assign a name to a number or string of characters. Please refer to the section on the "Defines Tab" in Chapter 3 of the LiveStage Professional User Manual for information on how to create your own custom defines.

By creating and using custom defines in your scripts, you can centralize the location for your custom constant values. For example:

In the Defines Tab:

```
kHomeURL = "http://www.totallyhip.com"
```

In the script:

```
GotoURL($kHomeURL)
```

The example above illustrates how defines are created and used. In the first part of the example, a define is added to the project in the Defines Tab to refer to a URL using the name "kHomeURL".

In the second part of the example, the custom define "kHomeURL" is used in a QScript Action that requires a string parameter.

The kHomeURL custom define can be used throughout the project. If you wish to change the value of the "kHomeURL" custom define, you simply change it in the

Defines Tab. Once you make the change in the Defines Tab, every instance where “kHomeURL” is used in your script will now use the new value.

To refer to a define, put a “$” in front of the its name. If the name of the define has spaces in it, then the name must be enclosed in quotes.

There are several defines created for you automatically. These are:

ThisSpriteID	Set to the ID of the sprite containing the script.
Images	Each image has a define created that represents the image index.
Custom Handlers	Each custom event handler has a define created that represents the Handler ID for that handler.

Built-in Defines

There are also built-in defines for various numeric constants used by QScript and QuickTime. You do not need to add a $ in front of Built-in Defines.

The following is a list of the QScript Built-in Defines. For more details on each of these built-in defines, please refer to the QScript Actions where they are used elsewhere in the QScript Reference.

Cursor IDs

```
kQTCursorOpenHand
kQTCursorClosedHand
kQTCursorPointingHand
kQTCursorRightArrow
kQTCursorLeftArrow
kQTCursorDownArrow
kQTCursorUpArrow
kQTCursorArrow
```

Drawing Modes Defines

```
srcCopy
srcOr
srcXor
```

srcBic
notSrcCopy
notSrcOr
notSrcXor
notSrcBic
blend
addPin
addOver
subPin
transparent
addMax
subOver
adMin
grayishTextOr
hilite
ditherCopy
graphicsModeStraightAlpha
graphicsModePreWhiteAlpha
graphicsModePreBlackAlpha
graphicsModeComposition
graphicsModeStraightAlphaBlend
graphicsModePreMulColorAlpha

Event IDs

kQTEventFrameLoaded
kQTEventIdle
kQTEventKey
kQTEventMouseClick
kQTEventMouseClickEnd
kQTEventMouseClickEndTriggerButton
kQTEventMouseEnter
kQTEventMouseExit
kQTEventMouseMoved
kQTEventListReceived
kQTEventMovieLoaded
kQTEventRequestToModifyMovie

Flash Mouse Transitions

kIdleToOverUp
kOverUpToIdle
kOverUpToOverDown
kOverDownToOverUp
kOverDownToOutDown
kOutDownToOverDown

kOutDownToIdle
kIdleToOverDown
kOverDownToIdle

Key Code Defines

kReturnKeyCode
kEnterKeyCode
kTabKeyCode
kBackSpaceKeyCode
kDeleteKeyCode
kInsertKeyCode
kUpArrowKeyCode
kDownArrowKeyCode
kLeftArrowKeyCode
kRightArrowKeyCode
kEscapeKeyCode
kPageUpKeyCode
kPageDownKeyCode
kHomeKeyCode
kEndKeyCode

Key String Defines (Use KeyIsDown or string comparisons only)

kReturnKey
kTabKey
kDeleteKey
kUpArrowKey
kDownArrowKey
kLeftArrowKey
kRightArrowKey
kDoubleQuoteCharacter
kLineFeedCharacter
kCRLFCharacters
kCRCharacter
kLFCharacter

Key Modifier Defines

kModifierActiveFlag
kModifierButtonState
kModifierCommandKey
kModifierShiftKey
kModifierAlphaLock
kModifierOptionKey
kModifierControlKey
kModifierRightShiftKey
kModifierRightOptionKey

kModifierRightControlKey

Key Modifier Flags (Used only with KeyIsDown)

kNone
kOptionKey
kShiftKey
kCommandKey
kControlKey
kCapsLockKey

Movie Looping Control Defines

kNoLoop
kLoop
kLoopPalindrome

Media Type Defines

kBaseMediaType
kFlashMediaType
kMPEGMediaType
kMusicMediaType
kQTVRQTVRType
kQTVRObjectType
kQTVROldPanoType
kQTVROldObjectType
kQTVRPanoramaType
kSpriteMediaType
kStreamingMediaType
kSkinMediaType
kSoundMediaType
kTextMediaType
kThreeDeeMediaType
kTimeCodeMediaType
kTweenMediaType
kVideoMediaType

Music Track Defines

kControllerModulationWheel
kControllerBreath
kControllerFoot
kControllerPortamentoTime
kControllerVolume
kControllerBalance
kControllerPan
kControllerExpression
kControllerLever1
kControllerLever2

kControllerLever3
kControllerLever4
kControllerLever5
kControllerLever6
kControllerLever7
kControllerLever8
kControllerPitchBend
kControllerAfterTouch
kControllerPartTranspose
kControllerTuneTranspose
kControllerPartVolume
kControllerTuneVolume
kControllerSustain
kControllerPortamento
kControllerSostenuto
kControllerSoftPedal
kControllerReverb
kControllerTremolo
kControllerChorus
kControllerCeleste
kControllerPhaser
kControllerEditPart
kControllerMasterTune
kControllerMasterTranspose
kControllerMasterVolume
kControllerMasterCPULoad
kControllerMasterPolyphony
kControllerMasterFeatures

Numeric Defines

EPSILON_FLOAT
MIN_FLOAT
MAX_FLOAT
MIN_LONG
MAX_LONG
MIN_SHORT
MAX_SHORT
MAX_USHORT
PI or π

QueryListServer Defines (QT5 or later)

kListQueryDebugTrace
kListQuerySendListAsKeyValuePairs
kListQuerySendListAsXML
kListQueryWantCallback

SendAppMessage Defines (QT5 or later)

kAppMsgDisplayChannels
kAppMsgRequestEnterFullScreen
kAppMsgRequestExitFullScreen
kAppMsgSoftwareChanged
kAppMsgWindowClose

State Defines

Movie Load States (QT5 or later)

kLoadStateError
kLoading
kLoaded
kPlayable
kPlaythroughOK
kComplete

Network States (QT5 or later)

kConnected
kNoNetwork
kNotConnected
kUncertain

Text Track Defines

Justification (QT5 or later)

kCenterJustify
kLeftJustify
kRightJustify

Text Display Flags (QT5 or later)

kDontDisplay
kDontAutoScale
kClipToTextBox
kUseMovieBGColor
kShrinkTextBoxToFIt
kScrollIn
kScrollOut
kHorizScroll
kReverseScroll
kContinuousScroll
kFlowHoriz
kContinuousKaraoke
kDropShadow
kAntiAlias
kKeyedText

kInverseHilite
kTextColorHilite

Text Edit States (QT5 or later)

kDirectEditing
kNoEditing
kScriptEditing

Text Styles (QT5 or later)

kNormalFace
kBoldFace
kItalicFace
kUnderlineFace
kOutlineFace
kShadowFace
kCondenseFace
kExtendedFace

Text Fonts (QT5 or later)

kAthensFont
kCairoFont
kCourierFont
kGenevaFont
kHelveticaFont
kLondonFont
kLosAngelesFont
kMobileFont
kMonacoFont
kNewYorkFont
kSanFranciscoFont
kSymbolFont
kTimesFont
kTorontoFont
kVeniceFont

Text Search Defines (QT5 or later)

kSearchAgain
kSearchCaseSensitive
kSearchCurrentSample
kSearchReverse
kSearchWraparound

Example

```
SpriteOfID($ThisSpriteID).SetVisible(TRUE)
SetImageIndexTo($"Up Arrow Image.qif")
ExecuteEvent($"Reset Game")
```

QT Version Various

See also Constants

Preprocessor Directives

Preprocessor Directives are added to scripts to control the compilation of a project.

#debug — Preprocessor Directives

Syntax ***#debug on | off***

on | off Indicates if debug output is to be turned on or off

Description This directive turns on or off the output of QScript source code to the Debugging Console.

Use this compiler directive to limit the amount of source code that is sent to the Debugging Console by selectively turning this directive on and off.

Example Use the following syntax to restrict the lines of script that are sent to the Debugging Console:

```
#debug on
LocalVars myNum
myNum = 100
       ...
#debug on
// Compute the value of myNum
       ...
```

```
#debug off
       ...
```

The above script sends only the script lines enclosed between the "`#debug on`" and "`#debug off`" statements to the Debugging Console.

#define — Preprocessor Directives

Syntax ***#define*** *defineName* **=** *defineValue*

defineName the name of the value definition

defineValue the value of the value definition

Description This directive creates a Value Definition that associates a name *(defineName*) with a value (*defineValue)*. Once defined, a Value Definition can be use anywhere within the scope of the current script.

Value Definitions are designed to work with the ***#include*** directive. Commonly used Value Definitions can be placed in a script file external to the LiveStage Professional Project and be included in the compilation process by the use of the ***#include*** directive. This allows you to develop script language segments that can be used in many projects.

To use a Value Definition in a QScript statement, add a $ to the front of the Value Definitions. For example:

```
#define kStartTime = 0
ThisMovie.GoToTime($kStartTime)
```

Example

Create Value Definitions in a file with the name of defines.qsf :

```
#define kMyURL = "http://www.totallyhip.com"
       ...
```

Add the following line to the beginning of any script where the Value Definitions in the file defines.qsf is to be used:

```
#include "defines.qsf"
GotoURL($kMyURL)
```

See also

#include

#include — Preprocessor Directives

Syntax

#include "*filename****"***

Description

This directive specifies a file containing additional QScript statements that are to be included at compilation time. This allows code segments to be placed in external files (external to the LiveStage Professional project) and shared between multiple projects.

Note: *The file specified must be saved in the Library folders (Global or Local) as this is the only place where these include files can be located.*

Example

```
#include "mycode.qsf"
```

See also

#define

Syntax ***#RegionFromImageFile(****filename****)***

filename A string literal specifying the name of the image file

Description This directive is used to specify a region defined using an image file. A black and white image file is supplied to provide the shape of the clipping region. A black and white image file must be used or the region created would be invalid.

Note: *The file specified must be saved in the Library folders (Global or Local) as this is the only place where these files can be located.*

Example

```
ThisTrack.SetClipRegionTo(
                #RegionFromImageFile(0,"myfile.pct")
// Sets the clipping region to the
// region specified by the image file
```

See also #RegionFromRect

#RegionFromRect — Preprocessor Directives

Syntax ***#RegionFromRect(****left, top, right, bottom****)***

left, top — Numeric literals specifying the coordinates for the top left corner of the rectangle

right, bottom — Numeric literals specifying the coordinates for the bottom right corner of the rectangle

Description This directive is used to specify a rectangular region. The parameters specify the location of the top left and bottom right corners of the region.

Example

```
ThisTrack.SetClipRegionTo(
                #RegionFromRect(0, 0, 100, 100))
// Sets the clipping region to the rectangle
// Specified
```

See also #RegionFromImageFile

#StringFromFile — Preprocessor Directives

Syntax ***#StringFromFile(****filename****)***

filename — A string literal specifying the name of the file

Description This directive specifies that a text file (specified) contains the text used in a string literal. The preprocessor directive is replaced with the contents of the specified file when the script is compiled.

Note: *The file specified must be saved in the Library folders (Global or Local) as this is the only place where these files can be located.*

Example

```
ReplaceText(#StringFromFile("mytext.txt"),
                1, GetTextLength)
// Replace the text in the current text sample
// with the characters in "mytext.txt"
```

See also #include

Targets

Most actions and properties require a target that specifies what is to execute the action or what is to return a property. If no target is specified, or a partial target is specified, then the default targets are used which are the current movie, track and sprite that is executing the script. Targets consist of three portions, the movie, the track within the movie and the sprite within the track. These can be completely or partially specified. You can specify just the sprite, or just the sprite and the track or just the track or just the movie. Whatever part of the target you do not specify will be replaced with the default target.

Event Target — Targets

Syntax ***`ThisEvent`***

Description

The ***`ThisEvent`*** target is used to access the temporary QTList accessible during the execution of an event handler. The temporary QTList attached to an event contains information specific to the event handler currently being executed.

The following are the layout of the event handler local QTLists for the various built-in events:

Key Pressed:

```
<event>
        <key>key value</key>
        <modifiers>key modifier flags</modifiers>
        <scancode>key scan code</scancode>
</event>
```

List Received:

```
<event>
       <listName>Name of the list</listName>
       <list>
              Contents of the returned list
                     ...
       </list>
</event>
```

Mouse Events (Click, Down, Enter, Exit, Moved, Up):

```
<event>
       <where>
              <h>X Position</h>
              <v>Y Position</v>
       </where>
</event>
```

All other events have empty temporary QTLists.

Example

```
LocalVar     theValue
SetString(theValue,
       ThisEvent.GetListElementValue("", 1))
// return the value of the first element in the
// event handler's temporary list
```

QT Version 5.0 or later

See also List Properties and Actions

Object Target — Targets

Syntax

```
ObjectNamed(name)
ObjectOfID(id)
ObjectOfIndex(index)
```

name — A string literal specifying the name of the object.

id — A numeric literal specifying the ID of the object.

index — A numeric literal specifying the index of the object.

Description — Objects within various Track (i.e. Sprite, QT3D, Flash, etc.) can be targeted by using its name, ID or index and corresponding Object target specifier.

Example

QT Version — 5.0 or later

See also

Movie Target — Targets

Syntax

```
MovieOfID(id)
MovieNamed(name)
ThisMovie
```

id — A numeric literal specifying the ID of the movie.

name — A string literal specifying the name of the movie.

Description — A movie can be targeted by using its ID or name to identify it. You can set the name or ID of the movie in the Info tab of the document window. In an html document you can also specify the name or ID of the move as part of the embed tag. You can do this using the free Plug-in Helper application from the following URL:

```
http://www.apple.com/quicktime/
                                developers/tools.html
```

or you can enter it yourself as follows:

```
<embed src="mymovie.mov" moviename="name" ...>
```

Example

```
MovieOfID (1)
MovieNamed ("My Movie")
```

QT Version 4.0 or later

See also

Movie Target (Parent Movies) — Targets

Syntax

RootMovie
ParentMovie

Description A child movie can use these to target the root movie, which is the movie that contains all other child movies. The parent movie is the movie that contains this child movie. Use these only from within a child movie.

Example

```
RootMovie.StartPlaying
ParentMovie.StartPlaying
```

QT Version 4.1 or later

See also Movie Target (Child Movies)

Movie Target (Child Movies) — Targets

Syntax

```
ChildMovieTrackNamed(name)
ChildMovieTrackOfID(id)
ChildMovieTrackOfIndex(index)
ChildMovieNamed(movie_name)
ChildMovieOfID(movie_id)
```

name — A string literal specifying the name of the movie track.

id — A numeric literal specifying the ID of the movie track.

index — A numeric literal specifying the index of the movie track.

movie_name — A string literal specifying the name of the child movie.

movie_id — A string literal specifying the ID of the child movie.

Description

These all target a child movie in some way. They do not target a track even though they mention tracks in their names. The movie that is targeted will be the currently loaded movie in the child movie track. If a movie is added to the child movie list of a track, but it is not loaded, then it can not be targeted because it does not exist yet. The targets that mention a child movie track all target the currently loaded movie in the child movie track specified. The targets that specify the movie name and movie ID will target a child movie that has the matching movie name or movie ID. This name or ID can be set for a movie in the info tab when you are creating the movie.

Example

```
ChildMovieTrackNamed("My Track").StartPlaying
ChildMovieNamed("My First Movie").StartPlaying
```

QT Version

4.1 or later

See also

Movie Target (Parent Movies)

QD3DObject Target — Targets

Syntax ***QD3DObjectNamed(****name****)***

name A string literal specifying the name of the QD3D Object.

Description A QD3D Objects can be targeted by using its name.

Example

QT Version 3.0 or later

See also

Sample Target — Targets

Syntax ***SpriteNamed(****name****)***

name A string literal specifying the name of the sample.

Description A media sample can be targeted by using its name. You can set the name of a media sample in the Sample Editor for each of the editable track types.

Example

```
LocalVars theTime
theTime = SampleNamed("MySample").StartTime
```

QT Version 4.0 or later

See also

Sprite Target — Targets

Syntax

```
SpriteOfID(id)
SpriteOfIndex(index)
SpriteNamed(name)
ThisSprite
```

id	A numeric literal specifying the ID of the sprite.
index	A numeric literal specifying the index of the sprite.
name	A string literal specifying the name of the sprite.

Description

A sprite can be targeted by using its ID, index or name to identify it. You can set the name and ID of the sprite from the Sprite Editor within LiveStage Professional. The index is set by LiveStage Professional.

Example

```
SpriteOfID(1).SetVisible(FALSE)
SpriteNamed("MySprite").SetVisible(TRUE)
```

QT Version 3.0 or later

See also

Track Target — Targets

Syntax

```
TrackOfIndex(index)
TrackNamed(name)
TrackOfID(id)
TrackOfType(type [, index] )
ThisTrack
```

index — A numeric literal specifying the index of the track.

name — A string literal specifying the name of the track.

id — A numeric literal specifying the ID of the track.

type — A numeric constant specifying the type of the track.

Description

A track can be targeted by using its index, ID, name or type to identify it. The index and name of a track is shown in the Tracks tab of the document window. The ID of the track is set by QuickTime when the movie is created and is not known until then, but it is usually the same as its index.

Example

```
TrackOfIndex(1)
TrackNamed("Picture Track 1")
TrackOfID(1)
TrackOfType(kVideoMediaType)
TrackOfType(kTextMediaType, 2)
```

QT Version

3.0 or later

See also

Functions

Functions are new to LiveStage Professional 3.0 and QuickTime 5. Functions perform tasks that are independent of other objects in a LiveStage Professional project. A function process values supplied via the parameter list and return a value. The following example demonstrates the use of Functions:

```
LocalVars theSquareRoot
theSquareRoot = Sqr(4)
```

In the above example, “Sqr(4)” calculates the square root of the parameter “4”. The result is assigned to the local variable “`theSquareRoot`”.

Functions are similar to Properties and can be used within expressions in place of numeric and string literals.

Math Functions

ArcTan — Math Functions

Syntax ***ArcTan****(value)*

value A numeric literal, variable or expression specifying the input value.

Description This function calculates the Trigonometric ArcTan of the supplied value.

Return Returns a numeric value in radians.

Example

```
LocalVars theValue
theValue = ArcTan(0.5)
```

QT Version 5.0 or later

See also ArcTan2, Cos, Sin, Tan

ArcTan2 — Math Functions

Syntax ***ArcTan2****(y, x)*

y A numeric literal, variable or expression specifying the y component or a rectangular vector.

x A numeric literal, variable or expression specifying the x component or a rectangular vector.

Description This function calculates the Trigonometric ArcTan2 of the supplied rectangular vector. A rectangular vector is supplied (y, x) which is used to return a full-range (-2 PI - 2 PI) angle value.

Return Returns a numeric value indicating the angle in radians calculated by the ArcTan2 function.

Example

```
LocalVars theValue
theValue = ArcTan2(10, 10)
```

QT Version 5.0 or later

See also ArcTan, Cos, Sin, Tan

Cos — Math Functions

Syntax ***Cos****(angle)*

angle A numeric literal, variable or expression specifying the input angle in radians.

Description This function calculates the Trigonometric Cosine of the supplied angle.

Return Returns a numeric value.

Example

```
LocalVars theValue
// Calculate the Cosine of 90 degrees
theValue = Cos(0.5)
```

QT Version 5.0 or later

See also ArcTan, Sin, Tan

DegreesToRadians — Math Functions

Syntax ***DegreesToRadians****(degrees)*

degrees A numeric literal, variable or expression specifying the input value in degrees.

Description This function converts the specified value in degrees to the equivalent value in radians.

Return Returns a numeric value.

Example

```
LocalVars theValue
// Convert 360 degrees to radians
theValue = DegreesToRadians(2.0 * PI)
```

QT Version 5.0 or later

See also RadiansToDegrees

Exp — Math Functions

Syntax ***Exp****(value)*

value A numeric literal, variable or expression specifying the input value.

Description This function calculates the Exponential of the specified value.

Return Returns a numeric value.

Example

```
LocalVars theExponent
// Calculates the exponential
theValue = Exp(100.0)
```

QT Version 5.0 or later

See also Log, Sqr

Log — Math Functions

Syntax ***Log****(value)*

value A numeric literal, variable or expression specifying the input value.

Description This function calculates the Natural Logarithm of the specified value.

Return Returns a numeric value.

Example

```
LocalVars theLog
// Calculates the Log
theValue = Log(100.0)
```

QT Version 5.0 or later

See also Exp, Sqr

RadiansToDegrees — Math Functions

Syntax ***RadiansToDegrees(****radians****)***

radians — A numeric literal, variable or expression specifying the input value in radians.

Description This function converts the specified value in radians to the equivalent value in degrees.

Return Returns a numeric value.

Example

```
LocalVars theValue
// Convert 32 PI radians to degrees
theValue = RadiansToDegrees(32.0)
```

QT Version 5.0 or later

See also DegreesToRadians

Random — Math Functions

Syntax ***Random(****minimum, maximum****)***

minimum A numeric literal or variable specifying the lowest number that can be returned.

maximum A numeric literal or variable specifying the highest number that can be returned.

Description This function generates and returns a random numeric value. The numeric value generated ranges from the minimum to the maximum values specified as parameters.

Note: *The Random function was reclassified a Math Function instead of a General Property for LiveStage Professional 3.0.*

Return Numeric value ranging from the specified minimum and maximum values.

Example

```
Random(1, 100)
// Generate a random number from 1 to 100
If (Random(1, 100) < 50)
// Script that will execute 50% of the time
EndIF
```

QT Version 3.0 or later

See also

Sin — Math Functions

Syntax ***Sin****(angle)*

angle A numeric literal, variable or expression specifying the input angle in radians.

Description This function calculates the Trigonometric Sine of the supplied angle.

Return Returns a numeric value.

Example

```
LocalVars theValue
// Calculate the Sine of 180 degrees
theValue = Sin(1.0)
```

QT Version 5.0 or later

See also ArcTan, Cos, Tan

Sqr — Math Functions

Syntax ***Sqr****(value)*

value A numeric literal, variable or expression specifying the input value.

Description This function calculates the Square Root of the specified value.

Return Returns a numeric value.

Example

```
LocalVars theSquareRoot
// Calculates the square root
theValue = Sqr(16)
```

QT Version 5.0 or later

See also Exp, Log

Tan — Math Functions

Syntax ***Tan****(angle)*

angle A numeric literal, variable or expression specifying the input angle in radians.

Description This function calculates the Trigonometric Tangent of the supplied angle.

Return Returns a numeric value.

Example

```
LocalVars theValue
// Calculate the Tangent of 45 degrees
theValue = Tan(0.25)
```

QT Version 5.0 or later

See also ArcTan, Cos, Sin

String Functions

StrLength — String Functions

Syntax ***StrLength****(string)*

string A string literal or variable specifying the input string.

Description This function returns the length of the specified string.

Return Returns a numeric value.

Example

```
LocalVars TheLength
// Calculates length of the string
TheLength = StrLength("This is a test")
```

QT Version 5.0 or later

See also StrCompare, StrConcat, SubString

StrCompare — String Functions

Syntax ***StrCompare****(string1, string2, caseSensitive, diacSensitive)*

string1 A string literal or variable specifying the first input string.

string2 A string literal or variable specifying the second input string.

caseSensitive A boolean literal or variable specifying if the strings are to be compared in a case sensitive manner.

diacSensitive A boolean literal or variable specifying if special handling for diacritical marks and special characters is to be used.

Description This function compares the two input strings and returns TRUE if the two strings are the same. Set *caseSensitive* to TRUE if you want to find an exact match. Set *diacSensitive* to TRUE if you are working with strings containing diacritical marks and special characters.

Return Returns a boolean value.

Example

```
LocalVars theSame
// Compares the two supplied strings
theSame = StrCompare("String1", "STRING1",
                     TRUE, FALSE)
// theSame contains FALSE as the two strings are
// not the same using case sensitive comparison
```

QT Version 5.0 or later

See also StrConcat, StrLength, SubString

StrConcat — String Functions

Syntax ***StrConcat(****string1, string2****)***

string1 A string literal or variable specifying the first input string.

string2 A string literal or variable specifying the second input string.

Description This function returns the concatenation of input strings.

Return Returns a string value.

Example

```
LocalVars theString
// Concatenate two strings
SetString(theString,
              StrConcat("Part 1 ", "Part 2"))
// theString now contains the string
// "Part 1 Part 2"
```

QT Version 5.0 or later

See also StrCompare, StrLength, SubString

SubString — String Functions

Syntax	***SubString(****string, offset, length****)***
string	A string literal or variable specifying the input string.
offset	A numeric literal or variable specifying the offset from the beginning of the string where the substring starts.
length	A numeric literal or variable specifying the length of the substring.
Description	This function returns a portion of the input string. The portion returned is controlled by the parameters *offset* and *length*.
Return	Returns a string containing the specified substring.

Example	`LocalVars theSub`
	`// Get the first 2 characters of the string`
	`SetString(theSub, SubString("Hi there?", 0, 2))`
	`// theSub contains "Hi"`

QT Version	5.0 or later
See also	StrCompare, StrConcat, StrLength

Properties and Actions

When you script in LiveStage Professional, you are working with QuickTime Objects such as Movies, Tracks, Samples, etc.

The way to access these QuickTime Objects is via Properties and Actions.

Properties

A property is a characteristic of an object. A car has a color, engine size, top speed. These are all properties of the car. Most properties of objects in QuickTime movies can only be changed by executing actions. They cannot be changed by assigning values directly to the properties.

Here is the syntax for accessing a Property of a QuickTime Object in QScript:

```
target.property [ (parameters...) ]
```

target	This is the specification indicating which QuickTime Object the property is attached to.
.	A . separates the two parts of the property access syntax. This is the standard syntax for object property access for contemporary scripting languages.
property	This is the name of the property that you want to access.
[*(parameters...)*]	Some properties accept parameters. These parameters must be placed between parentheses at the end of the property access statement.

Here are some examples of property access statements:

```
ThisMovie.MovieTime     // Accesses the current movie time
SpriteOfIndex(1).ID     // Accesses the sprite ID of the first sprite
```

You can use a property in any place that you would normally use a number. For example:

```
LocalVars theMovieTime
// Save the current movie time in a local variable
theMovieTime = ThisMovie.MovieTime
// Goto the latest downloaded time in the movie
ThisMovie.GotoTime(ThisMovie.MaxLoadedTimeInMovie)
```

You can also use them in expressions. For example:

```
If (ThisMovie.MovieTime < ThisMovie.MaxLoadedTimeInMovie)
// Go to the latest downloaded time in the movie
// if we are not there already
EndIf
```

Actions

Action are commands issued to QuickTime Objects to direct them to perform a task. Some Actions accept information in the form of parameters. Actions do not return a result so they can not be used where you would normally use a number or a property.

Here is the syntax for accessing an Action of a QuickTime Object in QScript:

target.*action* [(*parameters...*)] [**min**(*number*) **max**(*number*) **wraparound**]

target	This is the specification indicating which QuickTime Object the Action is attached to.
.	A "." separates the two parts of the action access syntax. This is the standard syntax for object action access for contemporary scripting languages.
action	This is the name of the action that you want to perform.
[(*parameters...*)]	Some actions accept parameters. These parameters must be placed between parentheses at the end of the property access statement.
[**min**(*number*) **max**(*number*) **wraparound**]	Some Actions can have their parameters limited to a range of values. This allows you to have a variable contain the value and not have to check it to see if it is in range. It also allows you to limit the range of relative actions (those that end in "by") so that the resultant setting falls within the range. To do this you need to add the optional **min** and **max** keywords after the action. The ranges should be enclosed in () and must be numeric literals or constants and cannot be variables or expressions. You can also add the **wraparound** keyword to any relative action so that when the resultant setting passes outside of its range it will then enter the range at the opposite end. Actions that can use any of these will show them in the syntax section of their descriptions.

General Properties and Actions

The general Properties and Actions do not require that a target be specified. These Properties and Actions accesses the properties and actions of the environment that is playing the movie.

General Properties

ComponentVersion	**General Properties**
Syntax	***ComponentVersion****(type, subtype, manufacturer)*
type	A string literal identifying the type of component.
subtype	A string literal identifying the sub-type of the component.
manufacturer	A string literal identifying the manufacturer of the component.
Description	This property returns the installed version number for the component of the given type, subtype and manufacturer. Each of the three parameters is a four character string literal.
Return	Returns a numeric value indicating the installed version of the component. Returns 0 if no component is installed.
Example	`If (ComponentVersion("fire", "", "Appl") > 2)`
QT Version	4.0 or later
See also	LoadComponent

ConnectionSpeed — General Properties

Syntax ***`ConnectionSpeed`***

Description This property returns the speed of the current internet connection in bits per second. A 14.4 modem speed would be returned as 1400 for instance.

Return A numeric value indicating the speed of the current internet connection in bits per second.

The following lists the connection speed reported for various connection settings:

Setting Value

14.4 Kbps1400
28.8/33.6 Kbps2800
56 Kbps Modem/ ISDN5600
112 Kbps ISDN/DSL11200
256 Kbps ISDN/DSL25600
384 Kbps ISDN/DSL38400
512 Kbps ISDN/DSL51200
768 Kbps IDSN/DSL76800
1 Mbps Cable100000
1.5 Mbps T1150000
Internet/LAN2147483648

Example

```
If (ConnectionSpeed < 28800)
```

QT Version 4.0 or later

See also

CustomHandlerID — General Properties

Syntax ***CustomHandlerID***

Description This property returns a unique custom action handler ID.

Return A numeric value representing a unique custom action handler ID.

Example

QT Version 5.0 or later

See also IsCustomHandlerOpen

IsCustomHandlerOpen — General Properties

Syntax ***IsCustomHandlerOpen***`(handler_ID)`

`handler_ID` A numeric value identifying the custom handler.

Description This property returns TRUE if the specified action handler is open otherwise it returns FALSE.

Return Returns a boolean value of TRUE if the handler is open otherwise it returns FALSE.

Example

QT Version 5.0 or later

See also CustomHandlerID

Syntax ***GetEventKey***

Description This property returns the key code of the key that triggered the KeyPressed Event.

Note: *This property should only be used within the KeyPressed Event handler otherwise the returned value may be invalid.*

Key Code Defines

```
kReturnKeyCode
kEnterKeyCode
kTabKeyCode
kBackSpaceKeyCode
kDeleteKeyCode
kInsertKeyCode
kUpArrowKeyCode
kDownArrowKeyCode
kLeftArrowKeyCode
kRightArrowKeyCode
kEscapeKeyCode
kPageUpKeyCode
kPageDownKeyCode
kHomeKeyCode
kEndKeyCode
```

For key codes not defined in the above list, simply refer to the key's ASCII value.

Note: *Do not use the Key String Defines with the GetEventKey property as these constants define strings instead of key codes.*

Return A numeric value indicating the key code when the KeyPressed event was triggered.

Example

```
LocalVars keyCode
keyCode = GetEventKey
```

QT Version 5.0 or later

See also GetEventMouseX, GetEventMouseY, GetEventModifiers, GetEventScanCode

GetEventModifiers — General Properties

Syntax

GetEventModifiers

Description

This property returns the key modifiers at the time the KeyPressed Event was triggered.

The following Modifier Key defines are available:

Modifier Define	Meaning
kModifierCommandKey	Command key held
kModifierShiftKey	Shift key held
kModifierAlphaLock	CapsLock key held
kModifierOptionKey	Option key held
kModifierControlKey	Control key held
kModifierRightShiftKey	Right Shift key held
kModifierRightOptionKey	Right Option key held
kModifierRightControlKey	Right Control key held

The above modifier defines can be used together to specify that two or more modifier keys are pressed at the same time (i.e. kModifierCommandKey + kModifierShiftKey)

Note: *This property should only be used within Mouse or Keyboard Event handler otherwise the returned value may be invalid.*

Return

A numeric value indicating the key modifier state when the Mouse or Keyboard event was triggered.

Example

```
If (GetEventModifiers = (kModifierCommandKey +
      kModifierShiftKey))
// Script executes only if the Command and Shift
// keys are pressed
EndIf
```

QT Version

5.0 or later

See also

GetEventMouseY, GetEventModifiers, GetEventKey, GetEventScanCode

GetEventMouseX — General Properties

Syntax ***GetEventMouseX***

Description This property returns the X coordinate of the mouse when the event was triggered. The coordinate returned is in the coordinate space of the track.

Return A numeric value indicating the X position of the mouse.

Example

```
LocalVars origMouseX
origMouseX = GetEventMouseX
```

QT Version 5.0 or later

See also GetEventMouseY, GetEventModifiers, GetEventKey, GetEventScanCode

GetEventMouseY — General Properties

Syntax ***GetEventMouseY***

Description This property returns the Y coordinate of the mouse when the event was triggered. The coordinate returned is in the coordinate space of the track.

Return A numeric value indicating the Y position of the mouse.

Example

```
LocalVars origMouseY
origMouseY = GetEventMouseY
```

QT Version 5.0 or later

See also GetEventMouseX, GetEventModifiers, GetEventKey, GetEventScanCode

GetEventScanCode — General Properties

Syntax ***GetEventScanCode***

Description This property returns the keyboard scan code of the key that triggered the KeyPressed Event.

Note: *This property should only be used within the KeyPressed Event handler otherwise the returned value may be invalid.*

Return A numeric value indicating the key scan code when the KeyPressed event was triggered.

Example

```
LocalVars scanCode
scanCode = GetEventScanCode
```

QT Version 5.0 or later

See also GetEventMouseX, GetEventMouseY, GetEventModifiers, GetEventKey

GetMemoryFree — General Properties

Syntax ***GetMemoryFree***

Description This property returns the number of bytes of free memory available.

Return A numeric value indicating the number of bytes of free memory available.

Example

```
LocalVars memoryFree
memoryFree = GetMemoryFree
```

QT Version 5.0 or later

See also

GetNetworkStatus — General Properties

Syntax ***GetNetworkStatus***

Description This property returns the current Network Status.

The following Network Status defines are available:

Network Status	Meaning
`kNoNetwork`	Network not available
`kNotConnected`	Network available but not connected
`kConnected`	Network available and connected
`kUncertain`	Unknown Network availability

Return A numeric value indicating the Network Status.

Example

```
If (GetNetworkStatus = kConnected)
// Launch URL only if already connect to
// to the network
GotoURL("http://www.totallyhip.com")
EndIf
```

QT Version 5.0 or later

See also

GetSystemVersion — General Properties

Syntax `GetSystemVersion`

Description This property returns the BCD (binary coded decimal) encoded system version number.

Return A numeric value indicating the BCD encoded system version number.

The following values are returned by various OS versions:

OS	Value Returned
MacOS 8.x	2144
MacOS 9.x	2308
MacOS X	0
Windows 9x	65536
Windows NT	131072

Example

QT Version 5.0 or later

See also Version

GMTDay — General Properties

Syntax `GMTDay`

Description This property returns the current day in GMT (Greenwich mean time). The day is returned as a number from 1 to 31. You can use this with the LocalDay property to determine what time zone the user is in.

Return A numeric value specifying the current day in GMT.

Example

```
LocalVars theDay
theDay = GMTDay
```

QT Version 4.0 or later

See also GMTMonth, GMTYear, LocalDay, LocalYear

GMTHours — General Properties

Syntax ***GMTHours***

Description This property returns the current hour in GMT (Greenwich mean time). The hour is returned as a number from 0 to 23. You can use this with the LocalHours property to determine what time zone the user is in.

Return A numeric value specifying the current hour in GMT.

Example

```
LocalVars localHours
localHours = GMTHours
```

QT Version 4.0 or later

See also GMTMinutes, GMTSeconds, LocalHours, LocalMinutes, LocalSeconds

GMTMinutes — General Properties

Syntax ***GMTMinutes***

Description This property returns the current minute in GMT (Greenwich mean time). The minute is returned as a number from 0 to 59. You can use this with the LocalMinutes property to determine what time zone the user is in.

Return A numeric value specifying the current minute in GMT.

Example

```
LocalVars localMins
localMins = GMTMinutes
```

QT Version 4.0 or later

See also GMTHours, GMTSeconds, LocalHours, LocalMinutes, LocalSeconds

GMTMonth — General Properties

Syntax ***GMTMonth***

Description This property returns the current month in GMT (Greenwich mean time). The month is returned as a number from 1 to 12. You can use this with the LocalMonth property to determine what time zone the user is in.

Return A numeric value specifying the current month in GMT.

Example

```
LocalVars theMonth
theMonth = GMTMonth
```

QT Version 4.0 or later

See also GMTDay, GMTYear, LocalDay, LocalMonth

GMTSeconds — General Properties

Syntax ***GMTSeconds***

Description This property returns the current second in GMT (Greenwich mean time). The second is returned as a number from 0 to 59.

Return A numeric value specifying the current seconds in GMT.

Example

```
LocalVars localSeconds
localSeconds = GMTSeconds
```

QT Version 4.0 or later

See also GMTHours, GMTMinutes, LocalHours, LocalMinutes, LocalSeconds

GMTYear — General Properties

Syntax ***GMTYear***

Description This property returns the current year in GMT (Greenwich mean time). The year is returned as a 4 digit number.

Return A numeric value specifying the current year in GMT.

Example

```
LocalVars theYear
theYear = GMTYear
```

QT Version 4.0 or later

See also GMTDay, GMTMonth, LocalDay, LocalMonth

HandlerRef — General Properties

Syntax ***HandlerRef***

Description This property returns the ID of the sprite that is executing this script. This is used when you create a sprite using the MakeNewSprite action. When you call the MakeNewSprite action you specify a sprite that contains the scripts for the new sprite. When those scripts are executed for the new sprite you can check this property to see what sprite is executing them.

Return Numeric value that is the ID of the sprite executing this script.

Example

```
SpriteOfID(HandlerRef).SetVisible(FALSE)
```

QT Version 4.0 or later

See also MakeNewSprite

IsRegistered — General Properties

Syntax ***IsRegistered(****version****)***

version — A numeric literal or variable indicating the QuickTime major version number to be tested (i.e. 5)

Description This property returns TRUE if the copy of QuickTime is registered (i.e. QuickTime Pro).

Return Boolean value indicating TRUE if the specified version of QuickTime Pro is being used.

Example

```
If (IsRegistered(5))
// Script executes only if QT Pro version 5.x
// is begin used
EndIf
```

QT Version 5.0 or later

See also Registered

KeyIsDown — General Properties

Syntax ***KeyIsDown(****modifiers, key****)***

modifiers — One of the modifier constants listed below.

key — A string literal containing the key to be tested.

Description This property returns a boolean value of TRUE if the keyboard key specified by the lowercase key is being held down. Modifiers is used to specify keyboard modifier keys (i.e. Shift key, Control key, etc.) that are held down in conjunction with the key specified by *key*. Extra modifiers can be pressed and TRUE will still be returned.

The following key string defines are available for specialized keys:

Key String Define

```
kReturnKey
kTabKey
kDeleteKey
```

```
kUpArrowKey
kDownArrowKey
kLeftArrowKey
kRightArrowKey
kDoubleQuoteCharacter
kLineFeedCharacter
kCRLFCharacters
kCRCharacter
kLFCharacter
```

Note: *Don't use these Key String Defines with the GetEventKey action. This action works with key codes and not key strings.*

The following modifier constants are available:

Key Modifier Flag	Meaning
`kNone`	No keyboard modifiers used
`kOptionKey`	Option key held down (Alt for Windows)
`kShiftKey`	Shift key held down
`kCommandKey`	Command key held down (Mac)
`kControlKey`	Control key held down
`kCapsLockKey`	Caps lock key held down

Note: *Don't use these Key Modifier Flags with the GetEventModifiers action. This action works with a different set of key modifier flags (see GetEventModifiers).*

Multiple modifier constants can be specified by separating them with the | operator.

Return

Boolean value of TRUE if the specified key is held down, FALSE otherwise.

Example

```
If (KeyIsDown(kNone, 'a') = TRUE)
// Script to execute if the "a" key is held down //
by itself
EndIf

If (KeyIsDown(kShiftKey, 'a') = TRUE)
// Script to execute if the "a" key is held down
// along with the shift key
EndIf

If (KeyIsDown(kShiftKey | kControlKey, 'a') = TRUE)
// Script to execute if the "a" key is held down
// along with the shift and control keys
EndIf
```

QT Version 3.0 or later

See also Boolean Literal, String Literal, Built-in Constants

LocalDay — General Properties

Syntax ***LocalDay***

Description This property returns the current day in local time. The day is returned as a number from 1 to 31.

Return A numeric value specifying the current day in local time.

Example

```
LocalVars localDayVar
localDayVar = LocalDay
```

QT Version 4.0 or later

See also GMTHours, GMTMinutes, LocalHours, LocalMinutes, LocalSeconds

LocalHours — General Properties

Syntax ***LocalHours***

Description This property returns the current hour in local time. The hour is returned as a number from 0 to 23.

Return A numeric value specifying the current hour in local time.

Example

```
LocalVars localHourVar
localHourVar = LocalHours
```

QT Version 4.0 or later

See also GMTMinutes, GMTHours, GMTSeconds, LocalMinutes, LocalSeconds

LocalMinutes — General Properties

Syntax ***LocalMinutes***

Description This property returns the current minute in local time. The minute is returned as a number from 0 to 59.

Return A numeric value specifying the current minute in local time.

Example

```
LocalVars localMinutesVar
localMinutesVar = LocalMinutes
```

QT Version 4.0 or later

See also GMTMinutes, GMTHours, GMTSeconds, LocalMinutes, LocalHours

LocalMonth — General Properties

Syntax ***LocalMonth***

Description This property returns the current month in local time. The month is returned as a number from 1 to 12.

Return A numeric value specifying the current month in local time.

Example

```
LocalVars localMonthVar
localMonthVar = LocalMonth
```

QT Version 4.0 or later

See also GMTDay, GMTMonth, GMTYear, LocalDay, LocalYear

LocalSeconds — General Properties

Syntax ***LocalSeconds***

Description This property returns the current second in local time. The second is returned as a number from 0 to 59.

Return A numeric value specifying the current second in local time.

Example

```
LocalVars localSecondsVar
localSecondsVar = LocalSeconds
```

QT Version 4.0 or later

See also GMTMinutes, GMTHours, GMTSeconds, LocalMinutes, LocalHours

LocalYear — General Properties

Syntax ***LocalYear***

Description This property returns the current year in local time. The year is returned as a 4 digit number.

Return A numeric value specifying the current year in local time.

Example

```
LocalVars localYearVar
localYearVar = LocalYear
```

QT Version 4.0 or later

See also GMTDay, GMTMonth, GMTYear, LocalDay, LocalMonth

MouseButtonDown — General Properties

Syntax ***MouseButtonDown***

Description This property returns a boolean value of TRUE indicating that the mouse button (left mouse button for Windows) is pressed, otherwise it returns FALSE.

Return Boolean value of TRUE if the mouse button is pressed, otherwise FALSE.

Example

```
If (MouseButtonDown)
// do something when button is pressed
EndIf
```

QT Version 3.0 or later

See also Boolean Literal

Platform — General Properties

Syntax ***Platform***

Description This property indicates what platform the movie is being run on. Currently only the Macintosh and Windows platforms are recognized.

Return A value of 1 is returned to indicate that the user is running on a Macintosh and a value of 2 is returned to indicate that they are running on a Windows machine.

Example

```
If (Platform = 1)
 // its a Mac
EndIf
```

QT Version 4.0 or later

See also

Registered — General Properties

Syntax ***Registered***

Description This property returns TRUE if QuickTime is registered. This means that the user has QuickTime Pro installed.

Return A boolean value of TRUE if QuickTime is registered otherwise FALSE is returned.

Example

```
If (Registered = TRUE)
 // do something
EndIf
```

QT Version 4.0 or later

See also IsRegistered, Version

Subscription — General Properties

Syntax ***Subscription(****channel_name****)***

channel_name A string literal identifying the channel by its URL.

Description This property returns TRUE if the named channel is currently subscribed to, otherwise it returns FALSE. Channel subscriptions show up in the pull out tray in the QuickTime player. Each channel is shown as an icon.

Return A boolean value of TRUE if the specified channel is subscribed to otherwise FALSE is returned.

Example

```
If (Subscription("My Test Channel") = TRUE)
```

QT Version 4.0 or later

See also AddSubscription, RemoveSubscription

TickCount — General Properties

Syntax ***TickCount***

Description This property returns the number of ticks (1/60 second) that have elapsed since the machine was started. You can use this for more accurate timing than you can get by using the idle handler.

Return A numeric value specifying how many ticks (1/60 second) have elapsed since the machine started up.

Example

```
x = TickCount + 5
While (x > TickCount)
// do something for 5 ticks
EndWhile
```

QT Version 4.0 or later

See also

Version — General Properties

Syntax ***Version***

Description This property returns the version of QuickTime installed. You can use this to determine if you can execute QScripts that require QuickTime 4 in order to work. You should only use greater than or less than comparisons so that your scripts will still work when a new version of QuickTime becomes available.

Return A numeric value indicating QuickTime version number.

Example

```
If (Version = 3)
// QuickTime 3
ElseIf (Version > 4)
// QuickTime version 4 or greater
EndIf
```

QT Version 4.0 or later

See also Registered

AddSubscription — General Actions

Syntax	***AddSubscription****(name, URL, pictures_URL)*
name	A string literal or variable containing the name of the channel.
URL	A string literal or variable containing the URL to the channel.
pictures_URL	A string literal or variable containing the URL to a picture for the channel.
Description	This action subscribes the user to the named channel with the given URL. The *name* parameter designates what name will be used for the subscription. The *URL* parameter refers to the media which the user is subscribing to. The *pictures_URL* parameter refers to a Gif image that is 33 x 28 pixels in size.
Example	`AddSubscription("Hip", "www.totallyhip.com", "www.totallyhip.com/channel.qif")`
QT Version	4.0 or later
See also	String Literal, RemoveSubscription, SetString, AppendString

AppendString — General Actions

Syntax ***AppendString(****Variable_1, Variable_2, Variable_Result****)***

Variable_1 A variable name as declared in a variable declaration.

Variable_2 A variable name as declared in a variable declaration.

Variable_Result A variable name as declared in a variable declaration.

Description This action sets the contents of the result variable to the concatenation of Variable_1 and Variable_2. All three parameters must be variable names.

Example

```
LocalVars firstString, secondString
LocalVars resultString
SetString(firstString, "I am ")
SetString(secondString, "Locutous of Borg.")
AppendString(firstString, secondString,
resultString)
```

In this case the result string will contain the string I am Locutous of Borg.

QT Version 4.0 or later

See also String Literal, SetString

ApplicationNumberAndString — General Actions

Syntax ***ApplicationNumberAndString(****number, string****)***

number A numeric expression.

string A string literal or variable.

Description This action sends the number and string to the application playing the movie.

Example The following statement causes the number 1 and the string "Here I am !" to be sent to the application playing the movie:

```
ApplicationNumberAndString(1, "Here I am!")
```

QT Version 3.0 or later

See also Numeric Expression, String Literal, DebugStr

CloseThisWindow — General Actions

Syntax ***CloseThisWindow***

Description This action sends a message to the application playing the movie (QuickTime Player) indicating that the current movie window is to be closed.

Example

QT Version 5.0 or later

See also DisplayChannels, EnterFullScreen, ExitFullScreen, SendAppMessage, SoftwareWasChanged

DebugStr — General Actions

Syntax ***DebugStr(****message****)***

message A string literal or variable.

Description This action sends a string to the application playing the movie. The parameter message contains the debug string to send. The string can be no longer than 255 characters.

Example The following statement causes the string "Hello World!" to be sent to the application playing the movie:

```
DebugStr("Hello World!")
```

QT Version 3.0 or later

See also String Literal, ApplicationNumberAndString

DisplayChannels — General Actions

Syntax ***DisplayChannels***

Description This action sends a message to the application playing the movie (QuickTime Player) indicating that the QuickTime TV channels should be displayed.

Example

QT Version 5.0 or later

See also CloseThisWindow, EnterFullScreen, ExitFullScreen, SendAppMessage, SoftwareWasChanged

EnterFullScreen — General Actions

Syntax ***EnterFullScreen***

Description This action sends a message to the application playing the movie (QuickTime Player) indicating that the current movie window is to enter full-screen mode.

Example

QT Version 5.0 or later

See also CloseThisWindow, DisplayChannels, ExitFullScreen, SendAppMessage, SoftwareWasChanged

ExecuteAppleScript — General Actions

Syntax ***ExecuteAppleScript(****string****)***

string A string literal or variable that contains the text of the AppleScript to execute.

Description This action is used only by LiveStage Professional at this time. With it you can create Tool Movies that can be placed in the Tools menu and used to control LiveStage Professional.

Example

```
ExecuteAppleScript("quit")
```

QT Version 4.0 or later

See also String Literal, Constants, SetString, AppendString

ExecuteGenericScript — General Actions

Syntax	***ExecuteGenericScript****(cmd, args)*
cmd	A string literal or variable containing the command.
args	A string literal or variable containing the arguments to be passed into the command being executed.
Description	It is up to the application that is playing the QuickTime movie to interpret this command.
Example	The following statement causes the command “DoAction” to be sent to the application playing the movie: `ExecuteGenericScript(“DoAction”, “argument”)`
QT Version	4.1 or later
See also	ExecuteVBScript, ExecuteJavaScript, ExecuteLingoScript

ExecuteJavaScript — General Actions

Syntax	***ExecuteJavaScript****(cmd, args)*
cmd	A string literal or variable containing the command.
args	A string literal or variable containing the arguments to be passed into the command being executed.
Description	It is up to the application that is playing the QuickTime movie to interpret this command.
Example	The following statement causes the command “DoAction” to be sent to the application playing the movie: `ExecuteJavaScript(“DoAction”, “argument”)`
QT Version	4.1 or later
See also	ExecuteGenericScript, ExecuteVBScript, ExecuteLingoScript

ExecuteLingoScript — General Actions

Syntax ***ExecuteLingoScript(****cmd, args****)***

cmd A string literal or variable containing the command.

args A string literal or variable containing the arguments to be passed into the command being executed.

Description It is up to the application that is playing the QuickTime movie to interpret this command.

Example The following statement causes the command DoAction to be sent to the application playing the movie:

```
ExecuteLingoScript("DoAction", "argument")
```

QT Version 4.1 or later

See also ExecuteGenericScript, ExecuteJavaScript, ExecuteVBScript

ExecuteProjectorScript. — General Actions

Syntax ***ExecuteProjectorScript(****cmd, args****)***

cmd A string literal or variable containing the command.

args A string literal or variable containing the arguments to be passed into the command being executed.

Description It is up to the application that is playing the QuickTime movie to interpret this command.

Example The following statement causes the command "DoAction" to be sent to the application playing the movie:

```
ExecuteProjectorScript("DoSoAction"argument")
```

QT Version 4.1 or later

See also ExecuteGenericScript, ExecuteJavaScript, ExecuteLingoScript

ExecuteVBScript — General Actions

Syntax ***ExecuteVBScript(****cmd, args****)***

cmd — A string literal or variable containing the command.

args — A string literal or variable containing the arguments to be passed into the command being executed.

Description It is up to the application that is playing the QuickTime movie to interpret this command.

Example The following statement causes the command "DoAction" to be sent to the application playing the movie:

```
ExecuteVBScript("DoAction", "argument")
```

QT Version 4.1 or later

See also ExecuteGenericScript, ExecuteJavaScript, ExecuteLingoScript

ExitFullScreen — General Actions

Syntax ***ExitFullScreen***

Description This action sends a message to the application playing the movie (QuickTime Player) indicating that the current movie window is to exit full-screen mode and return to the window mode.

Example

QT Version 5.0 or later

See also CloseThisWindow, DisplayChannels, EnterFullScreen, SendAppMessage, SoftwareWasChanged

Syntax ***GoToURL(****URL****)***

URL A string literal or variable that contains the URL to go to.

Description The GoToURL action is used to get content from either a Web Address (Internet/LAN) or local storage (CD-ROM or hard disk, etc.). Although its name indicates that its primary purpose is for Web Addresses there is a great deal of functionality incorporated into this action for non Web based operations.
By using the GoToURL action you can have the default system Web Browser display the specified URL, target a frame in your Web Page, launch the QuickTime (c) Player, etc. One caveat regarding this action is that all file specifications must be fully qualified (i.e. no referential paths).

This example will launch the default Web Browser on the target system and display the contents of the Totally Hip Web Site.

```
GoToURL("http://www.totallyhip.com")
```

You can also target a specific frame in the Web Browser (if it is already running) by using the following syntax:

```
GoToURL("<URL>T<Frame>")
```

Where Frame is the frame name.

To have QuickTime open the URL in a new window use the following GoToURL call.

```
GoToURL("<URL>T<_blank>")
```

A variation on the above tells QuickTime to open the URL in the QuickTime Player.

```
GoToURL("<URL>T<QuickTimePlayer>")
```

Another variation tells QuickTime to open the URL in the Default Web Browser.

```
GoToURL("<URL>T<webbrowser>")
```

To reference a file using the GoToURL action you would enter a call using a file specification. The syntax for the call is:

```
GoToURL("file:///<filename>")
```

Where filename is the platform specific file path. On the Mac this is volume:folder:file and on Windows it is volume:\folder\file.

Example

The following statement causes the default Web Browser to be launched and selects www.totallyhip.com to be the current URL.

```
GotoURL("http://www.totallyhip.com")
```

The following statement causes the specified URL (www.totallyhip.com) to be loaded into the specified Frame (SideBar):

```
GotoURL("<http://www.totallyhip.com>T<SideBar>")
```

QT Version

3.0 or later

See also

String Literal, SetString, AppendString, SetStatusString

LoadComponent — General Properties

Syntax ***LoadComponent****(component)*

component A numeric literal indicating the ID of the component to load.

Description This action causes the specified component to be loaded by QuickTime. This action initiates the loading of the specified QuickTime component. ComponentVersion should be called at a later time to ensure that the specified component is loaded.

This action allows the management of the automatic component download process provided by QuickTime 5.

Example

QT Version 5.0 or later

See also ComponentVersion

RemoveSubscription — General Actions

Syntax ***RemoveSubscription****(URL)*

URL A string literal or variable containing the URL to the channel.

Description This action removes the subscription from the specified URL. The user must be subscribed to the specified URL already, otherwise no action is performed.

Example

```
RemoveSubscription("http://www.totallyhip.com")
```

QT Version 4.0 or later

See also String Literal, AddSubscription

SendAppMessage — General Actions

Syntax ***SendAppMessage*****(***message***)**

message An integer expression indicating the application message ID to send.

Description This action sends the specified message ID to the application playing the movie.

Available application message IDs

```
kAppMsgSoftwareChanged
kAppMsgRequestWindowClose
kAppMsgRequestExitFullScreen
kAppMsgDisplayChannels
kAppMsgRequestEnterFullScreen
```

Note: *All of the above message IDs causes the QuickTime Player to perform the specified task. If you are not using QuickTime Player (QT 5 or later) to play the movie, your results will vary depending on the level of support offered by your player application.*

Example

```
SendAppMessage(kAppMsgRequestWindowClose)
// Tells QuickTime Player to close
```

QT Version 5.0 or later

See also DisplayChannels, EnterFullScreen, ExitFullScreen, SoftwareWasChanged, WindowThisClose

SetCursor — General Actions

Syntax ***SetCursor(****cursor_ID****)***

cursor_ID — ID of the cursor to use.

Description This action sets the current cursor. The available cursor IDs are as follows:

```
kQTCursorOpenHand
kQTCursorClosedHand
kQTCursorPointingHand
kQTCursorRightArrow
kQTCursorLeftArrow
kQTCursorDownArrow
kQTCursorUpArrow
kQTCursorArrow
```

Example

```
SetCursor(kQTCursorPointingHand)
```

QT Version 4.0 or later

See also String Literal, Constants, SetString, AppendString

SetStatusString — General Actions

Syntax ***SetStatusString(****status, Flags****)***

status — A string literal or variable that contains the status string.

Flags — Flags for the status string.

Description This action instructs the QuickTime plug-in to display the status string in the status area of the browser. The flags should be set to kStatusURL to indicate that this status string is a URL link. You can set it to kStatusStreaming if the string is a streaming status.

Example

```
SetStatusString("Loading...", kStatusURL)
```

QT Version 4.0 or later

See also String Literal, Constants, SetString, AppendString

SetString — General Actions

Syntax ***SetString(****Variable, String****)***

Variable — A variable name as declared in a variable declaration.

String — A string literal or numeric expression that will be converted to a string.

Description This action sets the contents of the variable to the provided string or numeric expression converted to a string.

Example

```
LocalVars myString
SetString(myString, "I am Locutous of Borg")
```

QT Version 4.0 or later

See also String Literal, AppendString, SetStatusString

SoftwareWasChanged — General Actions

Syntax ***SoftwareWasChanged***

Description This action sends a message to the application playing the movie (MoviePlayer) indicating that the installed QuickTime software has been updated.

Example

QT Version 5.0 or later

See also CloseThisWindow, DisplayChannels, EnterFullScreen, ExitFullScreen, SendAppMessage

Movie Properties and Actions

Movie properties and actions require a movie target. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. See the section on targets for a complete description and some examples of how to specify a target.

Movie Properties

GetDuration — Movie Properties

Syntax ***GetDuration***

Description This property returns the duration of the movie in the time scale of the movie (typically 600 units per second).

Return A numeric value representing the duration of the movie.

Example

```
LocalVars movieLengthInSeconds
movieLengthInSeconds = GetDuration / GetTimeScale
// Calculate the length of the movie in seconds
```

QT Version 5.0 or later

See also GetTimeScale

GetHeight — Movie Properties

Syntax ***GetHeight***

Description This property returns the height of the movie in pixels.

Return A numeric value representing the height of the movie.

Example

QT Version 5.0 or later

See also GetWidth

GetLoadState — Movie Properties

Syntax ***GetLoadState***

Description This property returns the current Load State of the movie.

The following defines are available for the different load states:

Load State Define	Meaning
`kLoading`	Movie is still loading
`kPlayable`	Movie is now playable
`kPlaythroughOK`	Movie has cached enough data to play through to the end
`kComplete`	Movie is completely loaded
`kLoadStateError`	Load error

Return A numeric value indicating the Load State of the movie.

Example

```
// Start playing the movie as soon as it is playable
If (GetLoadState = kPlayable AND MovieRate <> 0)
       StartPlaying
EndIf
```

QT Version 5.0 or later

See also MaxLoadedTimeInMovie

GetID — Movie Properties

Syntax ***GetID***

Description This property returns the ID of the movie. The movie ID of a movie can be set in the Info tab of LiveStage Professional.

Return A numeric value indicating the ID of the movie.

Example

```
LocalVars movieID
movieID = ThisMovie.GetID
// Retrieve the ID of the movie
```

QT Version 5.0 or later

See also GetName

GetName — Movie Properties

Syntax ***GetName***

Description This property returns the name of the movie. The movie name of a movie can be set in the Info tab of LiveStage Professional.

Return A string value indicating the name of the movie.

Example

```
LocalVars movieName
movieName = ThisMovie.GetName
// Retrieve the name of the movie
```

QT Version 5.0 or later

See also GetID

GetTimeScale — Movie Properties

Syntax ***GetTimeScale***

Description This property returns the Time Scale of the movie. This value is typically 600 units per second. The value of all movie and track durations are scaled by the Time Scale of the movie.

For Example, a duration of 4 seconds in a movie with the Time Scale of 600 would be represented as follows:

Scaled Duration = 4 seconds * 600 units per second = 2400

Return A numeric value representing the Time Scale of the movie.

Example

```
LocalVars movieLengthInSeconds
movieLengthInSeconds = GetDuration / GetTimeScale
// Calculate the length of the movie in seconds
```

QT Version 5.0 or later

See also GetDuration

GetTrackCount — Movie Properties

Syntax ***GetTrackCount***

Description This property returns the number of tracks in the movie.

Return A numeric value indicating the number of tracks in the movie.

Example

```
LocalVars trackCount
// Disable all tracks in the movie
For trackCount = 1 to GetTrackCount
TrackOfIndex(trackCount).SetEnabled(FALSE)
Next
```

QT Version 5.0 or later

See also

GetVariable — Movie Properties

Syntax ***GetVariable(****Address****)***

Address A numeric expression specifying the address of a movie variable.

Description Gets the value of the variable that is located at the specified address.

Movie variables (those variables created using MovieVars) are stored within their own sprite track, nothing else is placed into this track. This track always has the name "Movie Variables" so that you can get and set variables in this track from another movie. This allows for data transfer from one movie to another.

The primary purpose of this command is to enable data transfer between movies. Variables that specify an address can be between the address range of 1 to 10,000. Non-addressed variables are stored from the address 10,001 and on.

To assign an address to a variable you would define it using the following form:

```
MovieVars variable_name : address
```

The variable name is any valid identifier for your variable, the address field will contain a number in the range of 1 — 10,000. This will ensure that variable is stored at that location. The following gives an example of an array of 50 items which is located at address 200, thus the full address range for the elements in the array would be 200 — 249.

```
MovieVars myArray[50]:200
```

Note that for more technical readers you may notice that you can have variables defined in such a way that you actually have different representations of your data. For example, you can define two variables that access the same address areas.

```
MovieVars firstArray[50]:200
MovieVars secondVar:220
```

The example above creates two variables, the first being an array containing 50 elements which are stored at address 200 on. The second variable points to address 220, thus referring to item 21 in the array.

Return A value representing the content of the movie variable specified.

Example

```
x = MoveNamed("Remote").GetVariable(200)
```

QT Version 3.0 or later

See also SetVariable

GetWidth — Movie Properties

Syntax ***GetWidth***

Description This property returns the width of the movie in pixels.

Return A numeric value representing the width of the movie.

Example

QT Version 5.0 or later

See also GetHeight

IsMovieActive — Movie Properties

Syntax ***IsMovieActive***

Description This property returns TRUE if the movie is active.

Return A boolean value indicating the active state of the movie.

Example

QT Version 5.0 or later

See also

MaxTimeLoadedInMovie — Movie Properties

Syntax ***MaxLoadedTimeInMovie***

Description This property returns an integer indicating the amount of the movie that has been downloaded so far. The value returned is in terms of the time scale of the movie (usually 600). To calculate the loaded time in seconds, divide the value returned by the time scale of the movie (usually 600).

Return Numeric value indicating the loaded time of the movie using the time scale of the movie.

Example

```
If (MaxTimeLoadedInMovie > SampleNamed("My
Sample").StartTime)
// Script executes only when the
// MaxTimeLoadedInMovie is greater than the
// start time of the Sample named "My Sample"
EndIf
```

QT Version 4.0 or later

See also MovieTime, GotoTime

MovieIsLooping — Movie Properties

Syntax ***MovieIsLooping***

Description This property returns a boolean value of TRUE if the movie is set to loop when it reaches the end otherwise it returns FALSE.

Return Boolean value of TRUE if the movie is set to loop otherwise returns FALSE.

Example

```
If (MovieIsLooping = TRUE)
// Script executes only when the movie is looping
EndIf
```

QT Version 3.0 or later

See also Boolean Literal, SetLoopingFlag

MovieLoopIsPalindrome — Movie Properties

Syntax ***MovieLoopIsPalindrome***

Description This property returns a boolean value of TRUE if the movie is set to loop in a palindrome mode otherwise it returns FALSE. Palindrome looping mode plays the movie to the end then plays it backwards until the beginning then repeats the cycle.

Return Boolean value of TRUE if the loop mode of the movie is Palindrome otherwise FALSE is returned.

Example

```
If (MovieLoopIsPalindrome = TRUE)
// Script executes only when loop mode is
// Palindrome
EndIf
```

QT Version 3.0 or later

See also Boolean Literal, SetLoopingFlag

MovieRate — Movie Properties

Syntax

```
MovieRate
```

Description

This property returns the current playback rate of a movie. The value returned is in terms of the normal playback speed of the movie. Negative values indicate that the movie is playing backward. The following lists the meaning of possible Movie Rate values.

Movie Rate	Meaning
1.0	Playing forward at normal speed.
0	Stopped.
-1.0	Playing backward at normal speed.
1.5	Playing forward at 1.5x normal speed.
-4.0	Playing backward at 4x normal speed.

Return

Signed numeric value indicating the current playback rate of the movie.

Example

```
LocalVars theRate
theRate = MovieRate
```

QT Version

3.0 or later

See also

SetRateTo, SetRateBy, StartPlaying, StopPlaying

MovieTime — Movie Properties

Syntax ***MovieTime***

Description This property returns an integer indicating the current time of the movie. The value returned is in terms of the time scale of the movie (usually 600). To calculate the movie time in seconds, divide the value returned by the time scale of the movie (usually 600).

Return Numeric value indicating the current time of the movie using the time scale of the movie.

Example

```
If (MovieTime > SampleNamed("My Sample").StartTime)
// Script executes only when MovieTime is later
// than the start time of the Sample
// named "My Sample"
EndIf
```

QT Version 3.0 or later

See also GotoTime

MovieVolume — Movie Properties

Syntax ***MovieVolume***

Description This property returns the current volume level of a movie. The value returned is between 0 and 255 with 0 indicating silence and 255 indicating maximum volume. The volume settings of the individual tracks are all proportional to this master volume setting.

Return Numeric value between 0 and 255.

Example

```
LocalVars theVolume
theVolume = MovieVolume
```

QT Version 3.0 or later

See also TrackVolume, SetMovieVolume

Movie Actions

GetMovieURL — Movie Actions

Syntax ***GetMovieURL(****string_var****)***

string_var The name of a variable to receive the string value for the URL.

Description This action retrieves the URL of the movie placing it in the variable specified by the parameter *string_var*.

Obsolete - use GetParentMovieURL instead.

Example

```
LocalVars theMovieURL
GetMovieURL(theMovieURL)
```

QT Version 4.0 or later

See also GotoURL, SetString, AppendString

GetParentMovieURL — Movie Actions

Syntax ***GetParentMovieURL(****string_var****)***

string_var This is the name of a string variable to receive the URL for the parent movie.

Description This action retrieves the URL of the parent movie that contains this child movie issuing the command. You can then use this string to build a new URL relative to the parent movie in order to load in other assets.

Example

```
LocalVars urlString
GetParentMovieURL(urlString)
```

QT Version 4.1 or later

See also GetRootMovieURL

GetRootMovieURL — Movie Actions

Syntax ***GetRootMovieURL(****string_var****)***

string_var This is the name of a string variable to receive the URL for the root movie.

Description This action retrieves the URL of the root movie. You can then use this string to build a new URL relative to the root movie in order to load in other assets.

Example

```
LocalVars urlString
GetRootMovieURL(urlString)
```

QT Version 4.1 or later

See also GetParentMovieURL

GoToBeginning — Movie Actions

Syntax ***GoToBeginning***

Description This action rewinds a movie to its beginning.

Example The following statement causes the current movie to be rewound to the beginning:

```
GoToBeginning
```

QT Version 3.0 or later

See also GoToEnd, GotoTime, MovieTime

GoToEnd — Movie Actions

Syntax ***GoToEnd***

Description This action goes to the end of the movie.

Example The following statement causes the current movie to go to the end:

```
GoToEnd
```

QT Version 3.0 or later

See also GoToBeginning, GotoTime, MovieTime

GoToTime — Movie Actions

Syntax ***GoToTime(****time****)***

Time A numeric expression indicating the time in 600ths of a second. You can enter times in your scripts as mm:ss.hhh, (minutes:seconds.fraction of a second in 600ths).

Description This action jumps to a new time in the movie. To specify a movie time of 2 seconds, you would have to specify 1200 (2 seconds x 600 units per second or 00:02.000).

Example The following statements cause the current movie to jump to the 10 second position:

```
GoToTime(6000)
GotoTime(0:10.0)
```

The following statements cause the current movie to jump to the 5.5 second position:

```
GoToTime(600 * 5.5)
GotoTime(0:5.300)
```

QT Version 3.0 or later

See also GotoBeginning, GotoEnd, MovieTime

GoToTimeByName — Movie Actions

Syntax ***GoToTimeByName****(name)*

Name A string literal specifying the name of a chapter in the movie.

Description This action causes a movie to jump to the time of a chapter mark with the specified name.

In order to use this action the movie must have been created with chapter marks.

Example The following statement causes the current movie to jump to the Introduction chapter mark:

```
GoToTimeByName("Introduction")
```

QT Version 3.0 or later

See also GotoTime, GotoBeginning, GotoEnd, MovieTime

MovieChanged — Movie Actions

Syntax ***MovieChanged***

Description This action causes a notification to be sent to the application playing the movie indicating that the movie has been changed.

This notification would cause QuickTime Player to update its state.

Example

QT Version 3.0 or later

See also

PopAndGotoLabeledTime — Movie Actions

Syntax ***PopAndGotoLabeledTime(****label****)***

Label A string literal specifying the label of a saved time.

Description This action sets the current time of the movie to the time specified by the label. No action is performed if there is no time that matches the specified label.

Example

```
PopAndGotoLabeledTime("Intro Sequence")
```

QT Version 3.0 or later

See also PushCurrentTime, PushCurrentTimeWithLabel, PopAndGotoTopTime

PopAndGotoTopTime — Movie Actions

Syntax ***PopAndGotoTopTime***

Description This action sets the current time of the movie to the last saved time.

Example

```
PopAndGotoTopTime
```

QT Version 3.0 or later

See also PushCurrentTime, PushCurrentTimeWithLabel, PopAndGotoLabeledTime

PushCurrentTime — Movie Actions

Syntax ***PushCurrentTime***

Description This action saves the current time of the movie. The time may be retrieved later with the PopAndGotoTopTime and PopAndGotoLabeledTime commands.

Example

```
PushCurrentTime
```

QT Version 3.0 or later

See also PushCurrentTimeWithLabel, PopAndGotoTopTime, PopAndGotoLabeledTime

PushCurrentTimeWithLabel — Movie Actions

Syntax ***PushCurrentTimeWithLabel(****label****)***

Label — A string literal specifying the label for this saved time.

Description — This action saves the current time of the movie and attaches a label to it so that it may be referred to by name at some later point.

Example

```
PushCurrentTimeWithLabel("Intro Sequence")
```

QT Version — 3.0 or later

See also — PushCurrentTime, PopAndGotoTopTime, PopAndGotoLabeledTime

SetLanguage — Movie Actions

Syntax ***SetLanguage(****language****)***

Language — A numeric expression indicating the language to use. Ranges from 0 (English) to 150 (Greenlandic).

Description — This action sets the current language of the movie. This will cause tracks with the same groups to have the track with the matching language selected and enabled. If no track with the correct language can be found then the language is not changed.

Example

```
SetLanguage(1) // set language to french
```

QT Version — 3.0 or later

See also — TrackEnabled

SetLoopingFlags — Movie Actions

Syntax ***SetLoopingFlags(****flags****)***

Flags A numeric constant specifying the new looping flags.

Description This action sets the loop mode of a movie. The following are possible loop mode settings and their associated define:

Define	Loop Mode Description
kNoLoop	Movie will play forward once and stop.
kLoop	Movie will play forward to the end and then rewind and repeat.
kLoopPalindrome	Movie will play forward to the end then play backward until the beginning is reached and then repeat (ping pong).

Example The following statement causes the current movie to play forever:

```
SetLoopingFlags(kLoop)
```

The following statement causes the current movie to play in a ping pong fashion indefinitely:

```
SetLoopingFlags(kLoopPalindrome)
```

QT Version 3.0 or later

See also Defines, MoveIsLooping, MovieLoopIsPalindrome

SetPlaySelection — Movie Actions

Syntax ***SetPlaySelection(****play****)***

Play — A boolean literal of TRUE to indicate that only the selection should play.

Description This action sets or clears the Play Selection mode of a movie. The play parameter is either TRUE or FALSE.

A value of TRUE indicates that the movie will play only the selection (see *SetSelection* and *SetSelectionByName*. A value of FALSE will clear the Play Selection mode and allow the entire movie to be played.

Example The following statements will cause the selection from 2 seconds to 10 seconds in the current movie to be played:

```
SetSelection(0:2.0, 0:10.0)
SetPlaySelection(TRUE)
StartPlaying
```

QT Version 3.0 or later

See also SetSelection, TogglePlaySelection

SetRateBy — Movie Actions

Syntax

SetRateBy `(rate)` ***<MIN(***number***) MAX(***number***) wraparound>***

`Rate`

A numeric expression indicating the change in rate for the movie.

Description

This action causes the playback speed of a move to be altered by the specified amount relative to the current playback rate. The parameter rate can be positive or negative (see SetRateTo above).

Example

The following statement causes the playback speed of the current movie to increase by 1 full normal speed forwards:

```
SetRateBy(1)
```

QT Version

3.0 or later

See also

SetRateTo, StartPlaying, StopPlaying, MovieRate

SetRateTo — Movie Actions

Syntax ***SetRateTo*** `(rate)` ***<MIN(****number****) MAX(****number****)>***

`Rate` A numeric expression indicating the new rate for the movie.

Description This action sets the playback speed of a movie. The parameter Rate supplies the desired playback rate of the movie. The following lists some possible rate values and their meaning:

Rate	Meaning
0	Movie is stopped
1	Normal speed forward
-1	Normal speed in reverse

Example The following statement causes the current movie track to play at normal speed forwards:

```
SetRateTo(1)
```

The following statement causes the current movie track to play at double speed in reverse:

```
SetRateTo(-2)
```

The following statement causes the current movie track to stop:

```
SetRateTo(0)
```

QT Version 3.0 or later

See also StartPlaying, StopPlaying, SetRateBy, MovieRate

Syntax ***SetSelection(****start, end****)***

Start A numeric expression indicating the start time of the selection.

End A numeric expression indicating the end time of the selection.

Description This action specifies a selection within a movie. The two parameters specify the start and end times of the selection. The time values are in terms of the time scale of the movie (see GoToTime above).

Example The following statements cause a selection from 2 seconds to 10 seconds to be marked as the selection in the current i.e.:

```
SetSelection(2 * 600, 10 * 600)
SetSelection(0:2.0, 0:10.0)
```

QT Version 3.0 or later

See also SetSelectionByName, GoToTime, SetPlaySelection, TogglePlaySelection

SetSelectionByName — Movie Actions

Syntax ***SetSelectionByName(****start, end****)***

Start — A string literal indicating the start time by specifying a chapter name.

End — A string literal indicating the end time by specifying a chapter name.

Description This action specifies a selection within a movie. The two parameters specify the chapter names of the beginning and end of the selection.

In order to use this action, the movie must have been created with chapter marks.

Example The following statement causes a selection from the beginning of the "Introduction" to the beginning of the "Chapter 4" to be marked as the selection in the current movie:

```
SetSelectionByName("Introduction", "Chapter 4")
```

QT Version 3.0 or later

See also SetSelection, SetPlaySelection, TogglePlaySelection

SetVariable — Movie Actions

Syntax ***SetVariable(****Address, Value****)***

Address A numeric expression specifying the address of a movie variable.

Value A numeric expression specifying the value to store.

Description Sets the value of the variable that is located at the specified address.

For detailed information on how variable address work you should refer to the GetVariable documentation.

The address parameter specifies an address value between 1 and 10,000. These are all of the movie variables in the movie.

The Value parameter contains the value that you want to store at the specified address.

Example

```
MovieNamed("My Movie").SetVariable(8, 32)
```

Sets the variable at address 8 to the value of 32.

QT Version 3.0 or later

See also GetVariable

SetVolumeBy — Movie Actions

Syntax ***SetVolumeBy*** *(volume)* ***<MIN(****number****) MAX(****number****) wraparound>***

Volume — A numeric expression from -256 to 256 indicating the change in the volume setting.

Description This action alters the volume level of the movie by the specified amount. The parameter Volume supplies the value with which the volume level is changed up (positive value) or down (negative value).

Example The following statement causes the volume level of the movie to increase by 25:

```
SetVolumeBy(25)
```

QT Version 3.0 or later

See also Numeric Expression, SetVolumeTo, MusicVolume

SetVolumeTo — Movie Actions

Syntax ***SetVolumeTo*** *(volume)* ***<MIN(****number****) MAX(****number****)>***

Volume — A numeric expression from -256 (off) to 256 (full volume) indicating the new volume setting.

Description This action sets the volume level of the movie. The volume level of each track can be adjusted separately. The volume level of the movie controls the overall volume setting. A setting of 0 or less will mute the sound. You can use negative volume settings to keep track of the previous volume setting.

Example The following statement causes the volume of the movie to be turned off:

```
SetVolumeTo(0)
```

QT Version 3.0 or later

See also Numeric Expression, SetVolumeBy, MusicVolume

StartPlaying — Movie Actions

Syntax ***StartPlaying***

Description This action causes the movie to start playing at normal speed. It is a shortcut for SetRateTo(1.0).

Example

```
StartPlaying
```

QT Version 3.0 or later

See also SetRateTo, StopPlaying, SetRateBy, MovieRate

StepBackward — Movie Actions

Syntax ***StepBackward***

Description This action steps a movie backward a pre-determined period of time. The movie will always step backward to the start time of the previous visual sample.

Example The following statement causes the current movie to be stepped backward:

```
StepBackward
```

QT Version 3.0 or later

See also StepForward, MovieTime

StepForward — Movie Actions

Syntax ***StepForward***

Description This action steps a movie forward a pre-determined period of time. The movie will always step forward to the start time of the next visual sample.

Example The following statement causes the current movie to be stepped forward:

```
StepForward
```

QT Version 3.0 or later

See also StepBackward, MovieTime

StopPlaying — Movie Actions

Syntax ***StopPlaying***

Description This action causes the movie to stop playing. It is a shortcut for SetRateTo(0).

Example

```
StopPlaying
```

QT Version 3.0 or later

See also SetRateTo, SetRateBy, StartPlaying, MovieRate

TogglePlaySelection — Movie Actions

Syntax ***TogglePlaySelection***

Description This action toggles the Play Selection mode of a movie. If the current Play Selection mode of a movie is TRUE, calling TogglePlaySelection will change this to FALSE and vice versa.

Example The following statements will cause the current movie to turn off the Play Selection mode:

```
SetPlaySelection(TRUE)
TogglePlaySelection
```

QT Version 3.0 or later

See also SetPlaySelection, SetSelection, SetSelectionByName

Track Properties and Actions

Track properties need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Track Properties are available for all track types within a movie.

Track Properties

GetDuration — Track Properties

Syntax `GetDuration`

Description This property returns the duration of the movie in the time scale of the movie (typically 600).

Note: *The* `ThisTrack` *target must be used in conjunction with the GetDuration property in order to obtain the duration of the track. By default, if no target is specified, GetDuration returns the duration of the movie.*

Return A numeric value representing the duration of the movie.

Example

```
LocalVars trackLen
trackLen = TrackOfIndex(1).GetDuration /
                              GetTimeScale
// Calculate the duration of track 1 in seconds
```

QT Version 5.0 or later

See also GetTimeScale

GetHeight — Track Properties

Syntax ***GetHeight***

Description This property returns the height of the Track.

Note: *An explicit track target (i.e. TrackOfIndex(1)) must be used in conjunction with the GetHeight property in order obtain the height of the track. By default, if no target is specified, GetHeight returns the width of the movie.*

Return A numeric value representing the height of the track.

Example

```
LocalVars trackHeight
// Get the height of track 1
trackHeight = TrackOfIndex(1).GetHeight
```

QT Version 5.0 or later

See also GetWidth

GetID — Track Properties

Syntax ***GetID***

Description This property returns ID of the track. The ID of a track is assigned when the movie is loaded and therefore cannot be set explicitly.

Return A numeric value indicating the ID of the track.

Example

```
LocalVars trackID
trackID = TrackOfIndex(1).GetID
// Retrieve the ID of track 1
```

QT Version 5.0 or later

See also GetTrackName

GetName — Track Properties

Syntax ***GetName***

Description This property returns name of the track. The name of a track can be set in the Property Window of the track in LiveStage Professional.

Return A string value indicating the name of the track.

Example

```
LocalVars trackName
trackName = TrackOfIndex(1).GetName
// Retrieve the name of track 1
```

QT Version 5.0 or later

See also GetID

GetWidth — Track Properties

Syntax ***GetWidth***

Description This property returns the width of the Track.

Note: *An explicit track target (i.e. TrackOfIndex(1)) must be used in conjunction with the GetWidth property in order obtain the width of the track. By default, if no target is specified, GetWidth returns the width of the movie.*

Return A numeric value representing the width of the track.

Example

```
LocalVars trackWidth// Get the width of the track
trackWidth = TrackOfIndex(1).GetWidth
```

QT Version 5.0 or later

See also GetHeight

TrackEnabled — Track Properties

Syntax ***TrackEnabled***

Description This property returns a boolean value of TRUE if the track is enabled otherwise FALSE is returned. Only enabled tracks are processed by QuickTime. If a visual track (i.e. sprite track, video track, picture track, etc...) is enabled, it is displayed in the window of the movie. If an audible track (i.e. MIDI track, Instrument track, etc...) is enabled, it is played when the movie is played.

Return Boolean value indicating TRUE if the track is enabled otherwise FALSE is returned.

Example

```
If (TrackOfID(2).TrackEnabled)
```

QT Version 3.0 or later

See also Boolean Literal, SetEnabled, ToggleEnabled

Track Actions

SetEnabled — Track Actions

Syntax ***SetEnabled****(enable)*

Enable A boolean expression resulting in TRUE to enable the track.

Description This action enables or disables a track. A disabled track will not draw, make sounds or have any wired actions executed.

Example The following statement enables the track:

```
TrackOfID(2).SetEnabled(TRUE)
```

QT Version 3.0 or later

See also ToggleEnabled, TrackEnabled

ToggleEnabled — Track Actions

Syntax ***ToggleEnabled***

Description This action disables the track if it enabled or enables it if it is disabled.

Example The following statement causes the state of the current track to be inverted:

```
ToggleEnabled
```

QT Version 3.0 or later

See also SetEnabled, TrackEnabled

Sample Properties and Actions

The sample properties must have the sample target specified. The sample must exist in the current document and must be in one of the built-in tracks types.

Sample Properties

EndTime — Sample Properties

Syntax	***EndTime***
Description	Returns the end time of the sample. This value is calculated only when the script is compiled. This means that you can not pass in a string value to get the end time of a sample while your script is running.
Example	`GoToTime(SampleNamed("My Sample").EndTime)`
QT Version	3.0 or later
See also	GotoTime, StartTime

StartTime — Sample Properties

Syntax	***StartTime***
Description	Returns the end time of the sample. This value is calculated only when the script is compiled. This means that you can not pass in a string value to get the start time of a sample while your script is running.
Example	`GoToTime(SampleNamed("My Sample").StartTime)`
QT Version	3.0 or later
See also	GotoTime, EndTime

Spatial Track Properties and Actions

Spatial Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Spatial Tracks are tracks that have a visual representation in the movie. Spatial Track Properties and Actions are available for all spatial track targets. The built-in spatial track types are as follows:

Color Track
Effect Track
Flash Track
Movie Track
Picture Track
Sprite Track
Text Track
Video Track
VR Track

Spatial Track Properties

CanBeFocus — SpatialTrack Properties

Syntax ***CanBeFocus***

Description This property returns a boolean value indicating `True` if the target track can accept keyboard focus.

New for QuickTime 5, a number of QuickTime Track types can process the KeyPressed event. These tracks are the Flash, Text and Sprite tracks.

By setting the "Accept Focus" property for these tracks, the KeyPressed event will be triggered when the user types on the keyboard.

The `CanBeFocus` property returns the "Accept Focus" state of the target track.

Returns Boolean value indicating `True` if the "Accept Focus" property is set for the target track.

Example

```
If (CanBeFocus = true)
// Do something if the track can receive focus
Endif
```

QT Version 5.0 or later

See also IsFocus, SetFocus, EatKeyPressedEvent

Syntax ***IsFocus***

Description This property returns a boolean value indicating `True` if the target track is the current keyboard focus.

New for QuickTime 5, a number of QuickTime Track types can process the KeyPressed event. These tracks are the Flash, Text and Sprite tracks.

The `IsFocus` property returns `True` if the target track has keyboard focus.

Returns Boolean value indicating `True` if the target track has the keyboard focus.

Example

```
If (IsFocus = true)
// Do something if the track has keyboard focus
Endif
```

QT Version 5.0 or later

See also CanBeFocus, SetFocus, EatKeyPressedEvent

MouseHorizontal — SpatialTrack Properties

Syntax ***MouseHorizontal***

Description This property returns the current horizontal coordinate of the mouse pointer in the coordinate space of the track. The returned value is in pixels. Each time the MouseHorizontal property is accessed, the actual current horizontal coordinate of the mouse pointer is returned. This means that the value returned can change while the script is executing. In order to ensure that the same MouseHorizontal return value is used throughout the script, first assign the value to a local variable before using it in calculations.

Returns Numeric value indicating the horizontal coordinate of the mouse pointer in pixels.

Example

```
LocalVars curMouseX
curMouseX = MouseHorizontal
```

QT Version 3.0 or later

See also MouseVertical

MouseVertical — SpatialTrack Properties

Syntax ***MouseVertical***

Description This property returns the vertical coordinate of the mouse pointer in the coordinate space of the track. The returned value is in pixels. Each time the MouseVertical property is accessed, the actual current vertical coordinate of the mouse pointer is returned. This means that the value returned can change while the script is executing. In order to ensure that the same MouseVertical return value is used throughout the script, first assign the value to a local variable before using it in calculations.

Returns Numeric value indicating the vertical coordinate of the mouse pointer in pixels.

Example

```
LocalVars curMouseY
curMouseY = MouseVertical
```

QT Version 3.0 or later

See also MouseHorizontal

TrackHeight — SpatialTrack Properties

Syntax ***TrackHeight***

Description This property returns a numeric value indicating the height in pixels of the track. If the track is a non-visual track then 0 is returned.

Returns Numeric value indicating the height of the track in pixels.

Example

```
LocalVars value
value = TrackHeight
```

QT Version 3.0 or later

See also TrackWidth, ResetMatrix, ScaleMatrixBy

TrackLayer — SpatialTrack Properties

Syntax ***TrackLayer***

Description This property returns a numeric value indicating the visual layer of the track. Tracks with a higher layer number are drawn first. This means that tracks with a higher number appear behind tracks with a lower layer number.

Returns Numeric value indicating the visual layer of a track.

Example

```
LocalVars value
// Get the track layer of the current track
value = TrackLayer
// Get the track layer of the specified track
value = TrackOfID(4).TrackLayer
```

QT Version 3.0 or later

See also SetLayerTo, SetLayerBy

TrackWidth — SpatialTrack Properties

Syntax ***TrackWidth***

Description This property returns a numeric value indicating the width in pixels of the track. If the track is a non-visual track then 0 is returned.

Returns Numeric value indicating the width of the track in pixels.

Example

```
LocalVars value
value = TrackWidth
```

QT Version 3.0 or later

See also TrackHeight, ResetMatrix, ScaleMatrixBy

Spatial Track Actions

EatKeyPressEvent — Spatial Track Actions

Syntax ***EatKeyEvent***

Description This action prevents the KeyPress event from being sent back to the application playing the movie.

New for QuickTime 5, a number of QuickTime Track types can process the KeyPressed event. These tracks are the Flash, Text and Sprite tracks.

This action is used with KeyPressed event handlers.

Example

```
EatKeyEvent
// Indicate that the key was handled
```

QT Version 5.0 or later

See also CanBeFocus, IsFocus, SetFocus

MoveMatrixBy — Spatial Track Actions

Syntax ***MoveMatrixBy(X,Y)***

X,Y An integer constant that specifies the amount to move the matrix by in the x and y plane.

Description This action moves the track by the amount specified by the X and Y parameters.

Example

```
MoveMatrixBy(10,10)
```

QT Version 4.0 or later

See also MoveMatrixBy, MoveMatrixTo, ResetMatrix, ScaleMatrixBy, SetMatrixTo, SetMatrixBy

MoveMatrixTo — Spatial Track Actions

Syntax ***MoveMatrixTo(X,Y)***

X,Y — An integer constant that specifies where to move the matrix to in the x and y plane.

Description This action moves the track to the position specified by the X and Y parameters. It will also reset all other matrix settings to normal, thus losing all scaling and rotation that may have been applied to the matrix.

Example

```
MoveMatrixTo(10,10)
```

QT Version 4.0 or later

See also MoveMatrixBy, MoveMatrixTo, ResetMatrix, RotateMatrixBy, ScaleMatrixBy, SetMatrixTo, SetMatrixBy

ResetMatrix — Spatial Track Actions

Syntax ***ResetMatrix***

Description This action resets the display matrix of the track which removes any Spatial transformations that have been applied to it. This is a useful way to start the matrix in a known state before applying rotations or other transformations to it.

Example

```
ResetMatrix
```

QT Version 4.0 or later

See also MoveMatrixBy, MoveMatrixTo, RotateMatrixBy, ScaleMatrixBy, SetMatrixTo, SetMatrixBy

RotateMatrixBy — Spatial Track Actions

Syntax	***RotateMatrixBy(****angle, xloc, yloc****)***
Angle	A numeric literal or expression that specifies the amount to rotate the matrix by.
xloc, yloc	A numeric literal or expression that specifies the center of rotation
Description	This action rotates the track by the amount specified around the point specified by *xloc, yloc*.
Example	`RotateMatrixBy(5, 0, 0)`
QT Version	5.0 or later
See also	MoveMatrixBy, MoveMatrixTo, ResetMatrix, ScaleMatrixBy, SetMatrixTo, SetMatrixBy

ScaleMatrixBy — Spatial Track Actions

Syntax	***ScaleMatrixBy(****X,Y****)***
X,Y	A numeric constant that specifies the amount to scale the matrix by in the x and y plane.
Description	This action scales the track by the amount specified by the X and Y parameters.
Example	`ScaleMatrixBy(2,0.5)`
QT Version	4.0 or later
See also	MoveMatrixBy, MoveMatrixTo, ResetMatrix, RotateMatrixBy, SetMatrixTo, SetMatrixBy

SetClipRegionTo — Spatial Track Actions

Syntax ***SetClipRegionTo(****region****)***

region — A region specification provided by using the #RegionFromImageFile or #RegionFromRect preprocessor directives.

Description This action sets the clipping region of the spatial track. The *region* parameter is supplied by the preprocessor directives ***#RegionFromImageFile*** or ***#RegionFromRect.*** For more information on these preprocessor directives, please refer to the section on Preprocessor Directives.

Example

```
ThisTrack.SetClipRegionTo(
                #RegionFromRect(0, 0, 100, 100))
// Sets the clipping region to the rectangle
// specified
```

QT Version 3.0 or later

See also #RegionFromImageFile, #RegionFromRect

SetFocus — Spatial Track Actions

Syntax ***SetFocus***

Description This action sets the keyboard focus to the target track.

New for QuickTime 5, a number of QuickTime Track types can process the KeyPressed event. These tracks are the Flash, Text and Sprite tracks.

Example

```
SetFocus
// Direct the keyboard focus to the this track
```

QT Version 5.0 or later

See also CanBeFocus, IsFocus, EatKeyPressedEvent

SetGraphicsModeBy — Spatial Track Actions

Syntax ***SetGraphicsModeBy(****Mode, Red_Color, Green_Color, Blue_Color****)*** **<MIN(***number***) MAX(***number***) wraparound>**

Mode — A constant indicating one of the graphic modes to use.

Red_Color — A numeric constant indicating the brightness of the red component.

Green_Color — A numeric constant indicating the brightness of the green component.

Blue_Color — A numeric constant indicating the brightness of the blue component.

Description — This action changes the graphic mode the track uses to render itself, by the specified amount. The parameter Mode is a predefined constant indicating the graphic mode to set (see SetGraphicsModeBy or see Appendix II for a list of the graphic mode constants).

The parameters Red_Color, Green_Color and Blue_Color are numeric constants specifying the amount each primary colors is to change by. You can not actually change the mode by a relative amount, only its red, green and blue components.

Example — The following statement changes the red, green and blue components of the graphic mode of the current track by 1000:

```
SetGraphicsModeBy(srcCopy, 1000, 1000, 1000)
```

QT Version — 3.0 or later

See also — Numeric Expression, SetGraphicsModeTo

Syntax ***SetGraphicsModeTo****(Mode, Red_Color, Green_Color, Blue_Color)* **<MIN(***number***), MAX(***number***) >**

Mode A constant indicating one of the graphic modes to use.

Red_Color A numeric constant indicating the brightness of the red component

Green_Color A numeric constant indicating the brightness of the green component.

Blue_Color A numeric constant indicating the brightness of the blue component.

Description This action sets the graphic mode the track uses to render itself. See the list below or Appendix II - Drawing Mode Reference.

Available Drawing Modes

```
srcCopy
srcOr
srcXor
srcBic
notSrcCopy
notSrcOr
notSrcXor
notSrcBic
blend
addPin
addOver
subPin
transparent
addMax
subOver
adMin
grayishTextOr
hilite
ditherCopy
graphicsModeStraightAlpha
graphicsModePreWhiteAlpha
graphicsModePreBlackAlpha
```

```
graphicsModeComposition
graphicsModeStraightAlphaBlend
graphicsModePreMulColorAlpha
```

Example The following statement causes the graphic mode to be set to transparent using black as the transparent color:

```
SetGraphicModeTo(transparent, 0, 0, 0)
```

QT Version 3.0 or later

See also Numeric Expression, SetGraphicsModeBy

SetLayerBy — Spatial Track Actions

Syntax ***SetLayerBy*** *layer* ***<MIN(****number****) MAX(****number****) Wraparound>***

layer A numeric expression indicating the amount to change the layer by.

Description This action changes the visual layer of a track by the amount specified. Tracks with a higher number layer will draw first. This means that they will appear behind tracks with a lower layer number.

Example

```
SetLayerBy(1) Min(10) Max(20)
```

QT Version 3.0 or later

See also Numeric Expression, SetLayerTo, TrackLayer

SetLayerTo — Spatial Track Actions

Syntax ***SetLayerTo*** *(layer)* **<MIN(***number***) MAX(***number***) >**

layer A numeric expression indicating the layer.

Description This action sets the visual layer of a track. Tracks with a higher number layer will draw first. This means that they will appear behind tracks with a lower layer number.

Example

```
SetLayerTo(1)
```

QT Version 3.0 or later

See also Numeric Expression,. SetLayerBy, TrackLayer

SetMatrixBy — Spatial Track Actions

Syntax ***SetMatrixBy(m1, m2, m3, m4, m5, m6, m7, m8, m9)***

m1... m9 Numeric literals specifying the values for the individual cells of a 3x3 matrix

Description This action increments the matrix of the Spatial Track using the 9 parameters supplied.

The 9 parameters are arranged in a 3x3 matrix as follows:

Col. 1	2	3
Row		
1 **m1**	**m2**	**m3**
2 **m4**	**m5**	**m6**
3 **m7**	**m8**	**m9**

Example

QT Version 3.0 or later

See also MoveMatrixBy, MoveMatrixTo, ResetMatrix, RotateMatrixBy, ScaleMatrixBy, SetMatrixTo

Syntax ***SetMatrixTo(m1, m2, m3, m4, m5, m6, m7, m8, m9)***

m1... m9 Numeric literals specifying the values for the individual cells of a 3x3 matrix

Description This action sets the matrix of the Spatial Track using the 9 parameters supplied.

The 9 parameters are arranged in a 3x3 matrix as follows:

Col. 1	2	3
Row		
1 **m1**	**m2**	**m3**
2 **m4**	**m5**	**m6**
3 **m7**	**m8**	**m9**

Example

```
SetMatrixTo( 1, 0, 0,
             0, 1, 0,
             0, 0, 1)
```

QT Version 3.0 or later

See also MoveMatrixBy, MoveMatrixTo, ResetMatrix, RotateMatrixBy, ScaleMatrixBy, SetMatrixBy

Flash Track Properties and Actions

Flash Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Flash Tracks are also Spatial Tracks as they have a visual representation within the movie. Thus the Spatial Track Actions and Properties are also available for Flash Track Targets (see the section on Targets).

GetFlashVariable — Flash Track Properties

Syntax	***GetFlashVariable****(path, name)*
path	A string specifying the path to the variable.
name	A string specifying the name of the variable.
Description	This property returns specified Flash Track variable.
	Use the path string of "" to specify the root path in the Flash Track.
Return	The value of the specified variable in the Flash Track.

Example	`MovieVars flashVar`
	`flashVar = GetFlashVariable("", "MyVar")`
	`// Get the Flash variable "MyVar"`

QT Version	5.0 or later
See also	SetFlashVariable

Flash Track Actions

GoToFrameNamed — Flash Track Actions

Syntax ***GoToFrameNamed*(*name*)**

Name A string literal indicating the name of the frame to go to.

Description This action goes to a labeled frame with the specified name in the Flash Track. This is done by setting the current time in the movie to the time of the named frame in the Flash Track.

Example

```
GoToFrameNamed("Intro")
```

QT Version 4.0 or later

See also GoToFrameNumber, MovieTime

GoToFrameNumber — Flash Track Actions

Syntax ***GoToFrameNumber*(*frame*)**

Frame An integer expression indicating the frame number to go to.

Description This action goes to the frame with the specified frame number in the Flash Track. This is done by setting the current time in the movie to the time of the indicated frame in the Flash Track.

***Note:** Flash frames are numbered starting with 0 so to go to the 10th frame, you must specify the number 9.*

Example

```
GoToFrameNumber(12)
```

This example goes to the 13th frame in the Flash Track.

QT Version 4.0 or later

See also GoToFrameNamed, MovieTime

PerformFlashClick — Flash Track Action

Syntax ***PerformFlashClick****(path, buttonID, transition)*

path A string literal or variable specifying the path to the button

buttonID A numeric literal or variable specifying the ID of the button.

transition A numeric literal or variable specifying the mouse transition type.

Description This action simulates a mouse click within the Flash Track. A simulated mouse click is sent to the specified button.

Transition specifies the mouse transition type to send with the simulated mouse click:

Mouse Transition Defines	Value
kIdleToOverUp	0
kOverUpToIdle	1
kOverUpToOverDown	2
kOverDownToOverUp	3
kOverDownToOutDown	4
kOutDownToOverDown	5
kOutDownToIdle	6
kIdleToOverDown	7
kOverDownToIdle	8

Use the path string of "" to specify the root path in the Flash Track.

Example

```
PerformFlashClick("",100, 128)
// Simulate a button click on button ID 100
// in the Flash track
```

QT Version 5.0 or later

See also GetFlashVariable, SetFlashVariable

SetFlashVariable — Flash Track Actions

Syntax	***SetFlashVariable(****path, name, value, focus****)***
path	A string literal or variable specifying the path to the variable.
name	A string literal or variable specifying the name of the variable.
value	A string literal or variable specifying the value to set the Flash Variable to.
focus	A boolean literal or variable specifying if the focus is to be changed.
Description	This action sets the specified Flash Track variable to the value in the parameters. Use the path string of " " to specify the root path in the Flash Track.
Example	`SetFlashVariable("", "MyVar", "1234", FALSE)` `// Set the Flash variable "MyVar" to "1234"`
QT Version	5.0 or later
See also	GetFlashVariable

SetPan — Flash Track Actions

Syntax	***SetPan(****X_Percent, Y_Percent****)***
X_Percent	An integer constant specifying the percent to pan the Flash Track along the horizontal plane.
Y_Percent	An integer constant specifying the percent to pan the Flash Track along the vertical plane.
Description	This action pans the Flash Track by the specified x and y percentages.
Example	`SetPan(25, 45)`
QT Version	4.0 or later
See also	SetZoom

SetZoom — Flash Track Actions

Syntax	***SetZoom(****Zoom_Factor****)***
Zoom_Factor	An integer constant specifying the amount to zoom the Flash Track by.
Description	This action zooms the Flash Track by the zoom factor.
Example	`SetZoom(10)`
QT Version	4.0 or later
See also	SetPan

Syntax	***SetZoomRect(****Left, Top, Right, Bottom****)***
Left	An integer constant specifying the left edge of the rectangle to zoom to.
Top	An integer constant specifying the top edge of the rectangle to zoom to.
Right	An integer constant specifying the right edge of the rectangle to zoom to.
Bottom	An integer constant specifying the bottom edge of the rectangle to zoom to.
Description	This action zooms the Flash Track to the specified rectangle.
Example	`SetZoomRect(10, 10, 110, 100)` This example will set a display area in the track that is offset 10 pixels in on the X and Y axis from the top left corner and is 100 pixels high and wide.
QT Version	4.0 or later
See also	SetZoom

Movie Track Properties and Actions

Movie Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Movie Tracks are also Spatial Tracks as they have a visual representation within the movie. Thus the Spatial Track Actions and Properties are also available for Movie Track Targets (see the section on Targets).

Movie Track Actions

AddChildMovie	**Movie Track Actions**
Syntax	***AddChildMovie****(id, URL)*
id	A numeric expression indicating the ID for this new child movie.
URL	A string or string variable containing the URL for the child movie.
Description	This action adds a new child movie to the child movie list for the movie track. This does not load the movie or start it downloading. You only need to do this once for any

particular URL. If you are not creating the URL dynamically in your script, then you should add the URL to the movie list when you create the movie track.

Example `AddChildMovie(1000, "www.xcom.com/test.mov")`

QT Version 4.1 or later

See also LoadChildMovie

LoadChildMovie — Movie Track Actions

Syntax ***LoadChildMovie(id)***

id The ID is a numeric expression and must match an ID of a previously loaded child movie, or one added to the movie list when the movie track was created.

Description This action causes the child movie that matches the ID to start loading. The movie will display and optionally start playing when enough of it has downloaded. You should not immediately start the movie playing since it will not have started downloading. You should use MaxLoadedTimeInMovie to determine when some of the movie has downloaded before starting it playing. Keep in mind that only one movie is loaded at a time. Once you load a movie, then the previously loaded movie is released and discarded.

Example `LoadChildMovie(1000)`

QT Version 4.1 or later

See also AddChildMovie

LoadChildMovieWithQTList — Movie Track Actions

Syntax ***LoadChildMovieWithQTList(****id, XML****)***

id A numeric expression indicating the ID of the child movie.

XML A string literal or variable containing an XML formated string specifying the new elements that are to be loaded.

Description This action loads the supplied *XML* string into the QTList of the specified child movie in the target movie track.

Example

```
LocalVars myXML
// Load myXML with XML formated text
TrackNamed("Movies").LoadChildMovieWithQTLIst
                                (1, myXML)
```

QT Version 5.0 or later

See also QTList

RestartAtTime — Movie Track Actions

Syntax ***RestartAtTime(****time, rate****)***

time A numeric literal indicating the in the child movie to start playing from.

rate A numeric literal or expression for the playback rate.

Description This action plays the target child movie starting at the specified time and rate.

Example

```
// Play the child movie from the beginning
ChildMovieNamed("MyChild").RestartAtTime(0, 1.0)
```

QT Version 4.1 or later

See also

Music Track Properties and Actions

Music Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Track Actions and Properties are also available for Music Track targets (see the section on Targets).

Music Track Actions

PlayNote	**Music Track Actions**
Syntax	***PlayNote****(Instrument, Delay, Pitch, Velocity, Duration)*
Instrument	The index of the instrument in the first sample of the target instrument track.
Delay	The Delay parameter specifies a delay before the note starts playing. This value is measured in the movies Time Scale. A standard movie has a time scale of 600 so specifying 600 for your delay will create a 1 second delay.
Pitch	The Pitch parameter specifies the note's frequency. Values ranging from 0 to 127 are accepted, a value of 60 will produce a middle C, 59 is a middle B, etc.
Velocity	The Velocity parameter specifies the volume the note is to be played at. Values for this parameter range from 0 (no sound) to 100 (full volume).

Duration

The Duration parameter specifies the length of time the note will play for. This parameter operates like the Delay parameter. It is measured in the movies Time Scale. Note that instruments that naturally decay may not be obviously affected by this value.

Description

This action plays a note using the specified Music Instrument. The parameters Instrument, Delay, Pitch, Velocity and Duration are all Numeric Expressions specifying how the note is to be played.

Musical instruments are added to a LiveStage Professional project through the Instrument Media Sample window. To add an instrument to the LiveStage Professional project, click on the Add Built-in button at the bottom of the Instrument Media Sample window. An instrument selection dialog will be displayed allowing you to selection a MIDI style instrument. You may also add digital audio clips as instruments by dragging them into the instruments list in the Instrument Media Sample window.

Example

The following statement causes the note Middle C to be played using the first musical instrument in the track named Instruments 1 at half volume.

```
TrackNamed("Instruments 1").PlayNote(1, 0, 60, 100,
1000)
```

QT Version

3.0 or later

See also

Numeric Expression, SetController

Syntax	***SetController(****Sample, Instrument, Delay, Controller, Value****)***
Sample	The index of the sample of the target instrument track that contains the instrument to be altered.
Instrument	The index of the instrument in the sample of the target instrument track that is to be altered.
Delay	The Delay parameter specifies a delay before the SetController action takes place. This value is measured in the movies Time Scale. A standard movie has a time scale of 600 so specifying 600 for your delay will create a 1 second delay.
Controller	A numeric constant indicating the controller to modify.
Value	A numeric expression. This value has a different meaning for each controller.
Description	Controllers are used to modify the playback of an instrument. They can adjust characteristics such as Pitch, Blend, Pan, Reverb, etc. The setting of a controller is performed by indicating the instrument to adjust, the controller you want to modify and the value to set that controller to. Controller values control items such as pitch, blend and reverb. The controller numbers used by QuickTime are mostly identical to the standard MIDI controller numbers. These are signed 8.8 values. The full range, therefore is — 128.00 to 127 + 127 / 128 (or 0x8000 to 0x7FFF). All controls default to zero except for volume and pan. Pitch bend is specified in fractional semitones, which eliminates the restrictions of a pitch bend range. It can be bent as much as desired, and whenever it is needed. The last 16 controllers

(113-128) are global controllers. Global controllers respond when the part number is given as 0, indicating the entire synthesizer.

List of Controller Definitions

```
kControllerModulationWheel
kControllerBreath
kControllerFoot
kControllerPortamentoTime
kControllerVolume
kControllerBalance
kControllerPan
kControllerExpression
kControllerLever1
kControllerLever2
kControllerLever3
kControllerLever4
kControllerLever5
kControllerLever6
kControllerLever7
kControllerLever8
kControllerPitchBend
kControllerAfterTouch
kControllerPartTranspose
kControllerTuneTranspose
kControllerPartVolume
kControllerTuneVolume
kControllerSustain
kControllerPortamento
kControllerSostenuto
kControllerSoftPedal
kControllerReverb
kControllerTremolo
kControllerChorus
kControllerCeleste
kControllerPhaser
kControllerEditPart
kControllerMasterTune
kControllerMasterTranspose
kControllerMasterVolume
kControllerMasterCPULoad
kControllerMasterPolyphony
kControllerMasterFeatures
```

Example

QT Version 3.0 or later

See also Numeric Expression, PlayNote

QD3D Object Properties and Actions

QD3D support in QuickTime is obsolete and may not be included in future releases of QuickTime subsequent to version 5.0. MacOS X does not support QD3D objects.

QD3D Object Actions

RotateTo	QD3D Object Actions
Syntax	***RotateTo(X, Y, Z)***
X,Y,Z	Numeric constants that specify the angles for each of the three dimensions.
Description	This action will rotate the object to a position in 3D space.
Example	`RotateTo(10.6, 0, 100)`
QT Version	4.0 or later
See also	TranslateTo, ScaleTo

ScaleTo — QD3D Object Actions

Syntax	***ScaleTo(****X, Y, Z****)***
X,Y,Z	Numeric constants that specify the scale factors for each of the three dimensions.
Description	This action scales the object in 3D space by the amount specified in X,Y,Z.
Example	`ScaleTo(0.5, 1, 1)` This example will scale the 3D object 50% on its width and maintain full size in the Y and Z axis.
QT Version	4.0 or later
See also	TranslateTo, RotateTo

TranslateTo — QD3D Object Actions

Syntax	***TranslateTo(****X, Y, Z****)***
X,Y,Z	Numeric expression specifying the 3D position of the object.
Description	This action moves the object to the specified location in 3D space.
Example	`TranslateTo(100, 100, 5)` This example will move the object to an X position of 100, Y position of 100 and the Z position of 5.
QT Version	4.0 or later
See also	ScaleTo, RotateTo

Sound Track Actions

Sound Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Track Actions are also available for Sound Track targets (see the section on Targets).

Sound Track Properties

GetBass	**Sound Track Properties**
Syntax	***GetBass***
Description	This property returns the current Bass setting of the target sound track.
Returns	Numeric value indicating the Bass setting of the sound track.
Example	`LocalVars theBass` `theBass = GetBass`
QT Version	5.0 or later
See also	SetBassTreble

GetTreble — Sound Track Properties

Syntax ***GetTreble***

Description This property returns current Treble setting of the target sound track.

Returns Numeric value indicating the Treble setting of the sound track.

Example

```
LocalVars theTreble
theTreble = GetTreble
```

QT Version 5.0 or later

See also SetBassTreble

TrackBalance — Sound Track Properties

Syntax ***TrackBalance***

Description This property returns the balance setting of the track. The value returned is between –128 and 128. The value of –128 indicates that the balance is set to the left channel only while a value of 128 indicates right channel only. A value of 0 indicates center balance.

Returns Numeric value indicating the balance setting of the track.

Example

```
LocalVars theBalance
theBalance = TrackBalance
```

QT Version 3.0 or later

See also SetBalanceTo, SetBalanceBy

TrackVolume — Sound Track Properties

Syntax ***TrackVolume***

Description This property returns the volume level of the track. The value returned is between 0 (silence) and 255 (full volume). This is different from MovieVolume since TrackVolume returns only the volume of the specified track.

Returns Numeric value indicating the volume level of the track.

Example

```
LocalVars theVolume
theVolume = TrackVolume
```

QT Version 3.0 or later

See also MovieVolume, SetVolumeTo, SetVolumeBy

SetBalanceBy — Sound Track Actions

Syntax ***SetBalanceBy*(*balance*)<*MIN*(*number*) *MAX*(*number*) *wraparound*>**

balance — A numeric expression from –256 (left) to 256 (right) indicating the change in sound balance from left to right for the track.

Description — This action alters the left to right balance of a track by the specified amount. The parameter balance supplies the amount the balance is to be altered by. A negative value will shift the balance to the left while a positive value will shift the balance to the right.

Example — The following statement causes the balance of the current track to shift to the left by 10%.

```
SetBalanceBy(-25)
```

QT Version — 3.0 or later

See also — Numeric Expression, SetBalanceTo, TrackBalance

SetBalanceTo — Sound Track Actions

Syntax ***SetBalanceTo(****balance****)***

balance — A numeric expression from –128 (left) to 128 (right) indicating the sound balance from left to right for the track.

Description This action sets the sound balance for the track. The balance can change from the left speaker at full volume and the right speaker off all the way to the left speaker off and the right speaker at full volume.

Example The following statement sets the left to right balance of the current track to the center.

```
SetBalanceTo(0)
```

QT Version 3.0 or later

See also Numeric Expression, SetBalanceBy, TrackBalance

SetBassTrebleBy — Sound Track Actions

Syntax ***SetBassTrebleBy(bass, treble)***
<MIN(*number****) MAX(****number****) wraparound>***

bass — A numeric expression from -128 to 128 indicating the amount to change the bass setting for the sound track by.

treble — A numeric expression from -128 to 128 indicating the amount to change the treble setting for the sound track by.

Description This action sets the Bass and Treble for the sound track.

Example The following statement sets the Bass of the sound track sample by 10.

```
TrackNamed("MySound").SetBassTrebleBy(10, 0)
// Increase the bass setting by 10
```

QT Version 5.0 or later

See also GetBass, GetTreble, SetBassTrebleTo

SetBassTrebleTo — Sound Track Actions

Syntax ***SetBassTrebleTo(****bass, treble****)***

bass — A numeric expression from -128 to 128 indicating the amount to change the bass setting for the sound track by.

treble — A numeric expression from -128 to 128 indicating the amount to change the treble setting for the sound track by.

Description This action sets the Bass and Treble for the sound track.

Example The following statement sets the Bass and Treble of the sound track to neutral.

```
TrackNamed("MySound").SetBassTrebleTo(0, 0)
```

QT Version 5.0 or later

See also GetBass, GetTreble, SetBassTrebleBy

SetVolumeBy — Sound Track Actions

Syntax ***SetVolumeBy(****volume****)<MIN(****number****) MAX(****number****) wraparound>***

Volume — A numeric expression from -256 to 256 indicating the change in the volume setting.

Description This action alters the volume level of the track by the specified amount. The parameter Volume supplies the value by which the volume level is changed up (positive value) or down (negative value).

Example The following statement causes the volume level of the current track to increase by 25:

```
SetVolumeBy(25)
```

QT Version 3.0 or later

See also Numeric Expression, SetVolumeTo, TrackVolume

Syntax ***SetVolumeTo*** *(volume)* ***<MIN(****number****) MAX(****number****)>***

Volume A numeric expression from -256 (off) to 256 (full volume) indicating the new volume setting.

Description This action sets the volume level of the track. The volume level of each track can be adjusted separately. The volume level of the movie controls the overall volume setting. A setting of 0 or less will mute the sound from the track. You can use negative volume settings to keep track of the previous volume setting.

Example The following statement causes the volume of the current track to be turned off:

```
SetVolumeTo(0)
```

QT Version 3.0 or later

See also Numeric Expression, SetVolumeBy, TrackVolume

Sprite Track Properties and Actions

Sprite Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Sprite Tracks are also Spatial Tracks as they have a visual representation within the movie. Thus the Spatial Track Actions and Properties are also available for Sprite Track Targets (see the section on Targets).

Sprite Track Properties

GetIdleFrequency	**Sprite Track Properties**
Syntax	***GetIdleFrequency***
Description	This property returns the idle frequency in movie time in 1/60 second increments.
Return	Numeric value indicating the idle frequency.

Example	`LocalVars idleFreq`
	`idleFreq = GetIdleFrequency`
	`// Get the current track's idle frequency`

QT Version	5.0 or later
See also	SetIdleFrequency

NumSprites — Sprite Track Properties

Syntax ***NumSprites***

Description This property returns the number of sprites in a sprite track. If the track specified is not a sprite track then 0 is returned.

Return Numeric value indicating the number of sprites in the specified sprite track.

Example

```
IF (NumSprites > 100)
// Script executes only when the number of sprites is
// greater than 100
EndIf
```

QT Version 3.0 or later

See also MakeNewSprite, NewImages, SpriteOfIndex

NumImages — Sprite Track Properties

Syntax ***NumImages***

Description This property returns the number of images in a sprite track. If the track specified is not a sprite track then 0 is returned. Images are used by sprites to render their visual appearance.

Return Numeric value indicating the number of sprites in the specified sprite track.

Example

```
// Set the image of the current sprite to the
// last image in the sprite track
SetImageIndexTo(NumImages)
```

QT Version 3.0 or later

See also SetImageIndexTo, SetImageIndexBy, NumSprites

SpriteAtPoint — Sprite Track Properties

Syntax `SpriteAtPoint(x, y)`

Description This property returns the ID of the sprite that is at the point specified in the Sprite Track. The ID of the first sprite that intersects this point (going from the topmost to the bottommost sprite) is returned.

Return Numeric value indicating the ID the sprite that corresponds to the supplied X,Y location.

Example

QT Version 5.0 or later

See also

MakeNewSprite — Sprite Track Actions

Syntax	***MakeNewSprite****(Sprite_ID, Handler_ID, Image_Index, Visible, Layer)*
Sprite_ID	An integer expression specifying the ID of the new sprite.
Handler_ID	An integer expression specifying the ID of the sprite that will be used to handle events for the new sprite.
Image_Index	An integer expression specifying the initial image index for the new sprite.
Visible	A boolean expression specifying the visible state of the new sprite.
Layer	An integer expression specifying the drawing layer for the new sprite.
Description	This action creates a new sprite with the given Sprite_ID. The Handler_ID parameter indicates the sprite id of the sprite that will contain the handlers for this sprite. Note that you cannot use a sprite that was created with MakeNewSprite as the handler sprite. When the new sprite needs to handle an event, such as a Mouse Click event, the sprite with the ID of Handler_ID will have its handler called to handle the event. You can use the property CustomHandlerID to get the ID of the new sprite that was actually clicked on.
Example	`MakeNewSprite(100, 1, 3, TRUE, 1)` The above example will create a single sprite with the ID of 100 which uses sprite at ID 1 as its handler. The sprite will use the image at index 3, is created visible and will be placed on layer 1.
QT Version	4.0 or later
See also	DisposeSprite, CustomHandlerID

DisposeSprite — Sprite Track Actions

Syntax ***DisposeSprite(****Sprite_ID****)***

Sprite_ID — An integer expression specifying the ID of the sprite to dispose of.

Description This action will dispose of the sprite that has the specified ID. The sprite must have been previously created with MakeNewSprite.

Example

```
DisposeSprite(100)
```

The above example will dispose of the sprite that has ID 100.

QT Version 4.0 or later

See also MakeNewSprite

SetIdleFrequency — Sprite Track Actions

Syntax ***SetIdleFrequency(****frequency****)***

frequency — An integer expression indicating the frequency of the idle events in 1/60 second increments.

Description This action will change the idle frequency of the sprite track. Idle Frequency determines how often the Idle Event handlers are called by QuickTime.

Example

```
SetIdleFrequency(1 * 60)
// Set the idle frequency of the track to once
// every second
```

QT Version 5.0 or later

See also

QTList Properties and Actions

A QTList is a flexible data storage mechanism that is attached to QuickTime Movies and Tracks.

QTList Actions and Properties must be explicitly targeted when working with QTLists attached to Tracks. The following targeting must be used when targeting Tracks:

```
TrackNamed
TrackOfIndex
TrackOfType
```

QTList Actions and Properties without an explicit target will default to refer to the QTList attached to the movie.

Please note that the target specifier ThisTrack does nothing when used with QTList Actions and Properties and will always target the QTList of the movie.

QTLists can be used to store a wide variety of hierarchical information with a List Enabled QuickTime Object. The data storage metaphor is similar to that of an XML document. In fact, you can load all or part of a List using XML documents.

Elements in a QTList are addressed using a path string. The format of the path string is as follows:

```
path_parent.path_child1.path_child2.path_child...
```

Each part of the path string is separated by a . character (similar to the Property and Action access syntax). Each part of the path string specifies a branch in the hierarchical structure of the list.

For example, to access a particular date in a calendar, you would use the following path:

```
year.2001.month.february.day.16
```

The above path string would be used to address a QTList Element for February 16, 2001. The List hierarchy for such a calendar list would look like this in XML format:

```
<year>
       <2001>
              <month>
                     <january>
                     </january>
                     <february>
                            <day>
                                   <16>Some Data</16>
                            </day>
                     </february>
                     ...
              </month>
       </2001>
       ...
</year>
```

The following is an example of how to create a List Element for February 14, 2001. In the following examples, we will assume that the track we are using has the index of 1:

```
TrackOfIndex(1).AddListElement("year.2001.month.february.day",
                               1, "14")
```

In the above example, the path string of `year.2001.month.february.day` specified the branch within the hierarchy of the QTList where the new element is to be added. The 1 specified as the index indicates that the new element will be the first element in this part of the hierarchy of the QTList. In the example hierarchy, the index for the 16th would have been 1 prior to the new element being added. This index changes to 2 after the element insertion.

Now that the new element has been added, you can now assign it a value:

```
TrackOfIndex(1).SetListElement("year.2001.month.february.day.14",
                                  "Valentine's Day")
```

Executing the above line of script will set the element corresponding to February 14, 2001 to Valentine s Day .

To retrieve the value of this element, use the following script:

```
LocalVars elementValue
elementValue = TrackOfIndex(1).GetListElementValue(
                                    "year.2001.month.february.day.14")
```

When the element is no longer needed, you can delete it using the following script:

```
elementValue = TrackOfIndex(1).RemoveListElements(
                                "year.2001.month.february.day", 1, 1)
```

QTList Elements can also be accessed via their index number. For instance, in the above example, To access the February element you can use the following path:

```
"year.2001.month[2]"
```

The [2] in the path designates the second element in the container element month. You can thus iterate through the elements of the container element with a FOR loop utilizing GetListElementCount to get the number of elements in the container path (in this case year.2001.month). By assembling a path string that is appended with [*x*] (where *x* is the index), you can step through each element.

QTLists can only be attached to List Enabled QuickTime Objects such as Movies and Tracks. QTList properties and actions must be implicitly targeted as they do not make use of default targets.

QTLists can be preloaded into the Movie and Tracks by utilizing the UI provided in the Info Tab of the Document Window and in the Advanced Tab of the Track Properties Dialogs.

GetListAsXML — QTList Properties

Syntax ***GetListAsXML(****path, start, end****)***

path — A string literal or variable specifying the list path to the element

start — A numeric literal or variable specifying the first index

end — A numeric literal or variable specifying the last index

Description This property returns an XML formatted string containing the List elements specified by the *path* starting at index *start* and ending with index *end*.

Note: *The* start *and* end *index numbers must be within the valid range of elements. Specifying an invalid index will cause an error.*

Return An XML formatted string containing the list elements specified.

Example

```
LocalVars xmlString
LocalVars numElmt
numElmt = TrackOfIndex(1).GetListElementCount(
       "year.2001.month.february.day")
xmlString = TrackOfIndex(1).GetListAsXML(
       "year.2001. month.february.day",
       1, numElemt)
// Get the XML string for all days in February
```

QT Version 5.0 or later

See also

GetListElementCount — QTList Properties

Syntax ***GetListElementCount(****path****)***

path — A string literal or variable specifying the list path to the element.

Description This property returns the number of elements in the List Element specified by the *path*.

Return A numeric value indicating the number of elements in the specified List Element.

Example

```
LocalVars numElmt
numElmt = TrackOfIndex(1).GetListElementCount(
              "year.2001.month.february.day")
// Get the number of days defined in February
```

QT Version 5.0 or later

See also

GetListElementName — QTList Properties

Syntax ***GetListElementName(****path, index****)***

path — A string literal or variable specifying the list path to the element.

index — A numeric literal or variable specifying the index within the specified element.

Description This property returns the name of element specified by the *path* and *index* parameters.

Return A string value containing the name of the specified list element.

Example

```
LocalVars elmtName
SetString(elmtName,
TrackOfIndex(1).GetListElementName(
```

```
                    "year.2001.month.february.day", 1)
// Get the name of the first day in February
```

QT Version 5.0 or later

See also

GetListElementValue — QTList Properties

Syntax ***GetListElementValue(****path****)***

path A string literal or variable specifying the list path to the element.

Description This property returns the value of the QTList Element specified by the *path*.

Return A string value indicating the value stored in the specified QTList Element.

Example

```
LocalVars elmtValue
SetString(elmtValue,
      TrackOfIndex(1).GetListElementValue(
            "year.2001.month.february.day.14"))
// Get the value stored in February 14, 2001
```

QT Version 5.0 or later

See also

QTList Actions

AddListElement — QTList Actions

Syntax	***AddListElement****(path, index, name)*
path	A string literal or variable indicating the list path to the parent element for the new element.
index	A numeric literal or variable specifying the index of the new element to be added.
name	A string literal or variable containing the name of the new element.
Description	This action adds a new list element at the specified location. If the element addition is an insertion, all subsequent elements with the same parent element are renumbered.
Example	
QT Version	5.0 or later
See also	

Syntax ***ExchangeList(****URL, path****)***

URL A string literal or variable indicating the URL to send the list to.

path A string literal or variable containing the list path to the parent element.

Description This action calls the specified *URL* and sends the list elements referred to by the parameter *path* as an XML formatted parameter string. For example:

```
ExchangeList("www.myurl.com/somecgi.cgi", "")
```

The above line of script will send the entire list converted to XML format to the specified URL as a parameter:

```
www.myurl.com/somecgi.cgi
?qtlist=<qtlist> ... XML list content
... </qtlist>
```

Upon return, a List Received event is triggered for the track which sent the ExchangeList action so that the received list can be processed.

When the List Received event is triggered by a returned list being sent by the server, the returned list is saved in the temporary list of the event handler. To target the event s local list, you must use the ***ThisEvent*** target specifier.

The format or the local list of the List Received event is as follows:

```
<event>
       <listName>Name of the list</listName>
       <list>
              Contents of the returned list
                     ...
       </list>
</event>
```

Note: *ExchangeList is an asynchronous action. This means that the result of the ExchangeList action will not be available immediately following the execution of the action. The completion of the ExchangeList action is signaled by the triggering of the List Received event in the target track.*

Example

```
// List Received Event Handler
MovieVars myList
SetString(myList, ThisEvent.GetListAsXML(
"event.list", 1,
ThisEvent.GetListElementCount("event.list"))
// Retrieve the returned list as an XML string
```

QT Version 5.0 or later

See also QueryListServer

Syntax	***LoadListFromXML****(path, start, XML)*
path	A string literal or variable indicating the list path to the parent element where the new elements from the XML string is to be loaded.
start	A numeric literal or variable specifying the index where the new elements from the XML string is to be loaded.
XML	A string literal or variable containing an XML formatted string specifying the new elements that are to be loaded.
Description	This action pastes the elements contained in the specified XML formatted string into the list. The *path* and *start* parameters specify the location where the new elements are to be inserted.
Example	
QT Version	5.0 or later
See also	SetListFromURL

Syntax ***QueryListServer(****URL, keyValuePairs, flags, path****)***

URL A string literal or variable indicating the URL to send the list to.

keyValuePairs A string literal or variable containing the key/value pairs to be sent as parameters to the *URL*.

flags A numeric expression containing the flags used to control the action taken.

path A string literal or variable containing the list path to the parent element that will be sent as an XML parameter to the *URL*.

Description This action calls the specified *URL* and sends the list elements referred to by the parameter *path* as an XML formatted parameter string. For example:

```
QueryListServer("www.myurl.com/somecgi.cgi",
              "command=some_command",
              kListQuerySendListAsXML +
              kListQuerySendListAsNameValuePairs,
              "")
```

The above line of script will send the *keyValuePairs* string as well as the entire list converted to XML format to the specified URL as parameters. Here is what the above action would generate as a complete URL from the supplied parameters:

```
www.myurl.com/somecgi.cgi?command=some_
command
&qtlist=<qtlist> ... XML list content
... </qtlist>
```

The available flags are as follows:

Flag	Meaning
`kListQuerySendListAsXML`	Send the list specified by *`path`* as a parameter
`kListQuerySendListAsKeyValuePairs`	Send the string specified by *`keyValuePairs`* as parameters
`kListQueryWantCallback`	Specify that a List Received event is to be triggered by returning data
`kListQueryDebugTrace`	Send the complete URL generated as a DebugStr to the application running the movie.

Note: *These flags can be used together simply by adding them together (e.g.* `kListQuerySendListAsXML + kListQueryWantCallback` *will cause the supplied list to be sent along with the URL. In addition, a List Received event will also be received).*

Also note neither keyValuePairs nor path are required. Omitting the flags ***`kListQuerySendListAsXML`*** and ***`kListQuerySendListAsKeyValuePairs`*** will simply call the specified URL with no parameters.

Upon return, a List Received event is triggered for the track which sent the QueryListServer action so that the received list can be processed.

When the List Received event is triggered by a returned list being sent by the server, the returned list is saved in the temporary list of the event handler. To target the event's local list, you must use the ***`ThisEvent`*** target specifier.

The format of the local list of the List Received event is as follows:

```
<event>
       <listName>Name of the list</listName>
       <list>
              Contents of the returned list
                     ...
       </list>
</event>
```

Note: *QueryListServer is an asynchronous action. This means that the result of the QueryListServer action will not be available immediately following the execution of the action. The completion of the QueryListServer action is signaled by the triggering of the List Received event in the target track (if one is requested by using the kListQueryWantCallback flag).*

Example

QT Version 5.0 or later

See also ExchangeList

RemoveListElement — QTList Actions

Syntax	***RemoveListElement****(path, start, end)*
path	A string literal or variable indicating the list path to the parent element where the target elements are located
start	A numeric literal or variable specifying the index of the first element to be removed.
end	A numeric literal or variable specifying the index of the last element to be removed.
Description	This action removes the specified elements from the specified list path.
Example	
QT Version	5.0 or later
See also	

ReplaceListFromXML — QTList Actions

Syntax	***ReplaceListFromXML****(path, XML)*
path	A string literal or variable indicating the list path to the parent element where the replacement operation is to take place.
XML	A string literal or variable containing an XML formatted string containing element replacements.
Description	This action replaces the elements within the specified parent element (specified by *path*) with element names that match those in the specified XML formatted string.
Example	
QT Version	5.0 or later
See also	

SetListElement — QTList Actions

Syntax ***SetListElement****(path, value)*

path A string literal or variable indicating the list path to the element.

value A string literal or variable containing the new value of the element.

Description This action sets the value of the specified QTList element.

Example

QT Version 5.0 or later

See also

SetListFromURL — QTList Actions

Syntax ***SetListFromURL****(URL, path)*

URL A string literal or variable containing the URL where the XML file is located.

path A string literal or variable indicating the list path to the parent element where the new elements from the XML file are to be loaded.

Description This action loads the elements contained in the specified XML file into the list. The path parameters specify the location where the new elements are to be inserted.

Note: *SetListFromURL is a synchronous action. This means that the action will complete its operation before proceeding to the next command in the script (unlike asynchronous actions like ExchangeList and QueryListServer).*

Example

```
SetListFromURL("www.totallyhip.com/my.xml", "")
// Load the XML file "my.xml" to the root
// of the movie's QTList
```

QT Version 5.0 or later

See also LoadListFromXML

Sprite Properties and Actions

Sprite properties and actions need to have a sprite, track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). If no sprite target is specified then the sprite executing the script is considered to be the target. You can specify a sprite target by using SpriteNamed(name), SpriteOfID(ID) or SpriteOfIndex(index). See the section on targets for a complete description and some examples of how to specify a target.

Track and Spatial Track Properties and Actions are also available for Sprite Track targets (see the section on Targets).

Sprite Properties

BoundsBottom — Sprite Properties

Syntax	***`BoundsBottom`***
Description	As a sprite is moved, rotated, stretched and skewed, the bounding rectangle of the sprite changes. BoundsBottom returns the bottom of that bounding rectangle.
Return	Numeric value indicating the bottom edge of the sprite.
Example	`If (BoundsBottom > 100)`
QT Version	3.0 or later
See also	MoveTo, MoveBy, BoundsLeft, BoundsRight, BoundsTop

BoundsLeft — Sprite Properties

Syntax ***BoundsLeft***

Description As a sprite is moved, rotated, stretched and skewed, the bounding rectangle of the sprite changes. BoundsLeft returns the left edge of that bounding rectangle.

Return Numeric value indicating the left edge of the sprite.

Example

```
If (BoundsLeft > 100)
```

QT Version 3.0 or later

See also MoveTo, MoveBy, BoundsRight, BoundsTop, BoundsBottom

BoundsRight — Sprite Properties

Syntax ***BoundsRight***

Description As a sprite is moved, rotated, stretched and skewed, the bounding rectangle of the sprite changes. BoundsRight returns the right edge of that bounding rectangle.

Return Numeric value indicating the right edge of the sprite.

Example

```
If (BoundsRight > 100)
```

QT Version 3.0 or later

See also MoveTo, MoveBy, BoundsLeft, BoundsTop, BoundsBottom

BoundsTop — Sprite Properties

Syntax ***BoundsTop***

Description As a sprite is moved, rotated, stretched and skewed, the bounding rectangle of the sprite changes. BoundsTop returns the top edge of that bounding rectangle.

Return Numeric value indicating the top edge of the sprite.

Example
```
If (BoundsTop > 100)
```

QT Version 3.0 or later

See also MoveTo, MoveBy, BoundsLeft, BoundsRight, BoundsBottom

FirstCornerX — Sprite Properties

Syntax ***FirstCornerX***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. FirstCornerX returns the current X coordinate of the first corner.

Return Numeric value indicating the X coordinate of the first corner.

Example
```
If (FirstCornerX < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

FirstCornerY — Sprite Properties

Syntax ***FirstCornerY***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. FirstCornerY returns the current Y coordinate of the first corner.

Return Numeric value indicating the Y coordinate of the first corner.

Example

```
If (FirstCornerY < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

FourthCornerX — Sprite Properties

Syntax ***FourthCornerX***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. FourthCornerX returns the current X coordinate of the fourth corner.

Return Numeric value indicating the X coordinate of the fourth corner.

Example

```
If (FourthCornerX < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

FourthCornerY — Sprite Properties

Syntax ***FourthCornerY***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. FourthCornerY returns the current Y coordinate of the fourth corner.

Return Numeric value indicating the Y coordinate of the fourth corner.

Example

```
If (FourthCornerY < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

GetSpriteName — Sprite Properties

Syntax ***GetSpriteName***

Description This property returns name of the sprite. The name of a sprite can be set in the Sprite Track Sample Editor Window in LiveStage Professional.

Return A string value indicating the name of the sprite.

Example

```
LocalVars spriteName
spriteName = ThisSprite.GetSpriteName
// Retrieve the name of the sprite
```

QT Version 5.0 or later

See also ID

ID — Sprite Properties

Syntax *ID*

Description This property returns a numeric value that uniquely identifies the sprite. This is the value that would be supplied to the Target specifier SpriteOfID. You can give your sprites any ids that you desire and can use the id to do special processing, like using the id to specify the pitch in the PlayNote action.

Return Numeric value indicating the ID of the sprite.

Example
```
SetImageIndexTo(ID)°
```

QT Version 3.0 or later

See also Targets, SpriteOfID

ImageIndex — Sprite Properties

Syntax *ImageIndex*

Description This property returns the index of the image being used by the sprite. Sprites use images to render their visual appearance. Setting the image index to refer to another image in the sprite track changes the appearance of the sprite.

Return Numeric value indicating the image index being used by the sprite.

Example
```
If (ImageIndex = 3)
```

QT Version 3.0 or later

See also SetImageIndexTo, SetImageIndexBy, NumImages

ImageRegistrationPointX — Sprite Properties

Syntax ***ImageRegistrationPointX***

Description This property returns the X coordinate of the registration point of the image used by the sprite. Each image in the sprite track has a registration point. This value is the value entered in the image property portion of the Images Tab in the Sprite Sample editor window.

Return Numeric value indicating the X coordinate of the registration point of the image being used by the sprite.

Example

```
If (ImageRegistrationPointX = 0)
```

QT Version 3.0 or later

See also MoveTo, MoveBy

ImageRegistrationPointY — Sprite Properties

Syntax ***ImageRegistrationPointY***

Description This property returns the Y coordinate of the registration point of the image used by the sprite. Each image in the sprite track has a registration point. This value is the value entered in the image property portion of the Images Tab in the Sprite Sample editor window.

Return Numeric value indicating the Y coordinate of the registration point of the image being used by the sprite.

Example

```
If (ImageRegistrationPointY = 0)
```

QT Version 3.0 or later

See also MoveTo, MoveBy

Index — Sprite Properties

Syntax ***Index***

Description This property returns the index of the sprite. The index is the order number of the sprite in the list of sprites in the sprite track. This is the value that would be supplied to the Target specifier SpriteOfIndex.

Return Numeric value indicating the index of the sprite in the sprite list of the sprite track.

Example

```
If (Index > 10)
```

QT Version 3.0 or later

See also Targets, SpriteOfIndex, NumSprites

IsVisible — Sprite Properties

Syntax ***IsVisible***

Description This property returns a boolean value of `TRUE` if the sprite is visible otherwise `FALSE` is returned. Sprites that are not visible still receive mouse events; they are just not drawn when they are not visible.

Return Boolean value of `TRUE` if the sprite is visible otherwise `FALSE`.

Example

```
If (IsVisible = TRUE)
```

QT Version 3.0 or later

See also SetVisible, ToggleVisible

Layer — Sprite Properties

Syntax ***Layer***

Description This property returns a numeric value indicating the visual layer of the sprite within the sprite track. Sprites with a higher layer number are drawn first. This means that sprites with a higher number appear behind sprites with a lower layer number.

Return Numeric value indicating the layer of the sprite.

Example
```
If (Layer < SpriteOfID(1).Layer)
```

QT Version 3.0 or later

See also SetLayerTo, SetLayerBy

SecondCornerX — Sprite Properties

Syntax ***SecondCornerX***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. SecondCornerX returns the transformed X coordinate of the second corner.

Return Numeric value indicating the X coordinate of the second corner.

Example
```
If (SecondCornerX < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

SecondCornerY — Sprite Properties

Syntax ***SecondCornerY***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. SecondCornerY returns the current Y coordinate of the second corner.

Return Numeric value indicating the Y coordinate of the second corner.

Example

```
If (SecondCornerY < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

ThirdCornerX — Sprite Properties

Syntax ***ThirdCornerX***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. ThirdCornerX returns the current X coordinate of the third corner.

Return Numeric value indicating the X coordinate of the third corner.

Example If (ThirdCornerX < 10)˚

QT Version 3.0 or later

See also ResetMatrix, Stretch

ThirdCornerY — Sprite Properties

Syntax ***ThirdCornerY***

Description The corners of a sprite that is in its natural drawing state are the top left corner (FirstCorner), top right corner (SecondCorner), bottom right corner (ThirdCorner), and bottom left corner (FourthCorner). Once the sprite has been rotated and distorted, these corners will move to new positions. ThirdCornerY returns the current Y coordinate of the third corner.

Return Numeric value indicating the Y coordinate of the third corner.

Example

```
If (ThirdCornerY < 10)
```

QT Version 3.0 or later

See also ResetMatrix, Stretch

ClickOnCodec — Sprite Actions

Syntax	***ClickOnCodec(****X, Y****)***
X	An integer expression indicating the X coordinate of the location to simulate a mouse click.
Y	An integer expression indicating the Y coordinate of the location to simulate a mouse click.
Description	This action causes the ripple codec to ripple as if the user clicked at the point specified by the parameters X and Y.
Example	The following statement passes the coordinate (10, 10) to the ripple codec: `ClickOnCodec(10, 10)`
QT Version	3.0 or later
See also	Numeric_Constant, PassMouseToCodec

ExecuteEvent — Sprite Actions

Syntax ***ExecuteEvent(****Handler_ID****)***

Handler_ID An integer constant specifying the ID of the handler to call.

Description This action causes the specified event handler of a sprite to be executed. There are constants defined for the built-in event handlers so that you can call them too. You can also use the pre-defined define instead of the handler ID like so:

```
ExecuteEvent($"My Handler")
```

This approach may also be used to refer to a custom event by name instead of its ID:

```
ExecuteEvent($"custom event name")
```

Example The following statement causes the current sprite to execute its custom event 1000:

```
ExecuteEvent(1000)
```

QT Version 3.0 or later

See also Numeric Expression, Defines

MoveBy — Sprite Actions

Syntax ***MoveBy(****X, Y****)***

X An integer expression indicating the X distance to move by.

Y An integer expression indicating the Y distance to move by.

Description This action causes a sprite to be moved by the specified amount. You can use this in the idle event handler to cause the sprite to steadily move around on the stage.

Example The following statement causes the current sprite to be moved towards the top left by 1 pixel:

```
MoveBy(-1, -1)
```

QT Version 3.0 or later

See also Numeric Expression, MoveTo

MoveTo — Sprite Actions

Syntax ***MoveTo(X, Y)***

X An integer expression indicating the X coordinate of the location to move the sprite to.

Y An integer expression indicating the Y coordinate of the location to move the sprite to.

Description This action causes the registration point of a sprite to be moved to the specified coordinates.

Example The following statement causes the current registration point of sprite to be moved to the coordinate (0, 0):

```
MoveTo(0, 0)
```

QT Version 3.0 or later

See also Numeric Expression, MoveBy

PassMouseToCodec — Sprite Actions

Syntax ***PassMouseToCodec***

Description When the mouse is clicked on a sprite, you can call this action to pass the mouse click on to the codec that is rendering the image for the sprite. If this is the ripple codec, this will cause a ripple effect to appear where the mouse was clicked.

Example The following statement when placed in the Mouse Click handler of a sprite will cause the ripple codec to ripple where the mouse was clicked:

```
PassMouseToCodec
```

QT Version 3.0 or later

See also ClickOnCodec

ResetMatrix — Sprite Actions

Syntax ***ResetMatrix***

Description This action causes the display matrix of the sprite to be reset, thus undoing any Spatial transformations that had been applied to it.

Example

```
ResetMatrix
```

QT Version 3.0 or later

See also Rotate, Scale

Rotate — Sprite Actions

Syntax ***Rotate(****Degrees****)***

Degrees An integer expression indicating the number of degrees to rotate the sprite by.

Description This action causes a sprite to be rotated by the specified number of degrees. You can only rotate the sprite to a specific angle by first resetting its matrix by calling ResetMatrix and then calling Rotate.

Example The following statement causes the current sprite to rotate by 45 degrees counter clockwise:

```
Rotate(-45)
```

QT Version 3.0 or later

See also Numeric Expression, ResetMatrix

Scale — Sprite Actions

Syntax	***Scale(****X, Y****)***
X	An integer expression indicating the amount to scale the sprite in the horizontal direction.
Y	An integer expression indicating the amount to scale the sprite in the vertical direction.
Description	This action causes a sprite to be scaled by the specified amounts. Scale values smaller than 1 specify that the sprite is to be shrunk while values greater than 1 enlarge the sprite.
Example	The following statement causes the current sprite to be expanded by 200%: `Scale(2, 2)`
QT Version	3.0 or later
See also	Numeric Expression, ResetMatrix

Syntax ***SetGraphicsModeBy(Mode, Red_Color, Green_Color, Blue_Color) <MIN(number) MAX(number) wraparound>***

Mode A constant indicating one of the graphic modes to use.

Red_Color A numeric constant indicating the brightness of the red component.

Green_Color A numeric constant indicating the brightness of the green component.

Blue_Color A numeric constant indicating the brightness of the blue component.

Description This action changes the graphic mode a sprite uses to render itself by the specified amount. The parameter Mode is a predefined constant indicating the graphic mode to set (see SetGraphicsModeBy see Appendix II for a list of the graphic mode constants).

The parameters Red_Color, Green_Color and Blue_Color are Numeric Expressions specifying the amount each primary colors is to change by.

Example The following statement changes the red, green and blue components of the graphic mode of the current sprite by 1000:

```
SetGraphicsModeBy(srcCopy, 1000, 1000, 1000)
```

QT Version 3.0 or later

See also Numeric Expression, SetGraphicsModeTo

Syntax ***SetGraphicsModeTo(Mode, Red_Color, Green_Color, Blue_Color)*** ***<MIN(****number****) MAX(****number****)>***

Mode A constant indicating one of the graphic modes to use.

Red_Color A numeric constant indicating the brightness of the red component

Green_Color A numeric constant indicating the brightness of the green component.

Blue_Color A numeric constant indicating the brightness of the blue component.

Description This action sets the graphic mode a sprite uses to render itself. The parameter Mode is a predefined constant indicating the graphic mode to set. See the list below or Appendix II - Drawing Mode Reference.

Available Drawing Modes

```
srcCopy
srcOr
srcXor
srcBic
notSrcCopy
notSrcOr
notSrcXor
notSrcBic
blend
addPin
addOver
subPin
transparent
addMax
subOver
adMin
grayishTextOr
hilite
ditherCopy
graphicsModeStraightAlpha
graphicsModePreWhiteAlpha
```

```
graphicsModePreBlackAlpha
graphicsModeComposition
graphicsModeStraightAlphaBlend
graphicsModePreMulColorAlpha
```

Example The following statement graphic mode to be set to transparent using black as the transparent color:

```
SetGraphicModeTo(transparent, 0, 0, 0)
```

QT Version 3.0 or later

See also Numeric Expression, SetGraphicsModeBy

SetImageIndexBy — Sprite Actions

Syntax ***SetImageIndexBy(index) <MIN(****number****) MAX(****number****) wraparound>***

Index A numeric expression indicating the amount to change the image index by for the sprite.

Description This action alters the image index of a Sprite by the specified amount. The parameter Index indicates the amount to modify the image index of a sprite by.

Images in a LiveStage project can be referenced by their Index numbers. Changing the image index of the sprite has the effect of changing its appearance since setting a different image index would cause a different image to be used in rendering the Sprite. SetImageIndexBy can be used to animate a Sprite using a series of images.

Example The following code fragment sets the current image index of the sprite to the next image in a series of 5 images. This example assumes that there is a Sprite and a series of 5 images in the LiveStage project with the index of 1 through 5.

In the Idle event handler of the Sprite enter the following QScript statements:

```
SetImageIndexBy(1) MIN(1) MAX(5) wraparound
```

Export and run the movie. The result is a movie that animates the 5 images that you have in the project in a continuous loop.

QT Version 3.0 or later

See also Numeric Expression, SetImageIndexTo, ImageIndex

SetImageIndexTo — Sprite Actions

Syntax ***SetImageIndexTo(****Index****)*** ***<MIN(****number****) MAX(****number****)>***

Index A numeric expression indicating the new image index to use for the sprite.

Description This action sets the image index of a Sprite. The parameter Index is indicates the new image index to set for the Sprite.

Images in a LiveStage project can be referenced by their Index numbers or by using their name, which is pre-defined to be its index. Changing the image index of a sprite has the effect of changing its appearance since setting a different image index would cause a different image to be used in rendering the Sprite. SetImageIndexTo can be used to animate a Sprite using a series of images.

Example The following code fragment sets the image index of the current sprite to the next image in a series of 5 images. This example assumes that there is a Sprite and a series of 5 images in the LiveStage project with the index of 1 through 5.

In the Frame Loaded event handler of the Sprite enter the following QScript statements:

```
GlobalVars CurrentCel
CurrentCel = 0
```

In the Idle event handler of the Sprite enter the following QScript statements:

```
GlobalVars CurrentCel
CurrentCel = CurrentCel + 1
SetImageIndexTo(CurrentCel) MIN(1) MAX(5)
```

Export and run the movie. The result is a movie that animates the 5 images that you have in the project and halts at the last image.

QT Version 3.0 or later

See also Numeric Expression, SetImageIndexBy, ImageIndex

SetLayerBy — Sprite Actions

Syntax ***SetLayerBy(****layer****) <MIN(****number****) MAX(****number****) Wraparound>***

layer A numeric expression indicating the amount to change the layer by.

Description This action changes the visual layer of a sprite by the amount specified. Sprites with a higher number layer will draw first. This means that they will appear behind sprites with a lower layer number.

Example

```
SetLayerBy(1) Min(10) Max(20)
```

QT Version 3.0 or later

See also Numeric Expression, layer, SetLayerTo

SetLayerTo — Sprite Actions

Syntax ***SetLayerTo*** `(layer)` ***<MIN(****number****) MAX(****number****)>***

`layer` A numeric expression indicating the layer.

Description This action sets the visual layer of a sprite. Sprites with a higher number layer will draw first. This means that they will appear behind sprites with a lower layer number.

Example

```
SetLayerTo(1)
```

QT Version 3.0 or later

See also Numeric Expression, Layer

SetVisible — Sprite Actions

Syntax **SetVisible(***Visible***)**

Visible A boolean expression that makes the sprite visible if `TRUE`.

Description This action shows or hides a Sprite. The Visible parameter is a Boolean_Expression indicating that the sprite should be shown (`TRUE`) or hidden (`FALSE`).

Example The following statement causes the current sprite to be hidden:

```
SetVisible(FALSE)
```

QT Version 3.0 or later

See also Boolean_Expression, IsVisible, ToggleVisible

Stretch — Sprite Actions

Syntax `Stretch(X1, Y1, X2, Y2, X3, Y3, X4, Y4)`

X1-X4 The x coordinate of each of the four corners of the sprite.

Y1-Y4 The y coordinate of each of the four corners of the sprite.

Description The Stretch action is a very powerful function that allows you to manipulate the shape of a sprite. The parameters you specify in the Stretch action represent the four corners of the sprite. These start with the top left, top right, bottom right and finally the bottom left.

This action causes a sprite to be skewed. The parameters X1, Y1 through X4, Y4 specifying Numeric Expressions containing a series of 4 points corresponding to the four corners of the sprite specifying the new location of the corners.

Although Stretch gives you a great deal of control of a sprites shape there are a few quirks in QuickTime which can make the stretch action operate in unexpected ways.

No Warped Corners Allowed.

Example

QT Version 3.0 or later

See also Numeric Expression, ResetMatrix

ToggleVisible — Sprite Actions

Syntax ***ToggleVisible***

Description This action toggles the visible state of a Sprite.

Example The following statements cause the current sprite to be shown and then hidden:

```
SetVisible(TRUE)
ToggleVisible
```

QT Version 3.0 or later

See also SetVisible, IsVisible

Text Track Properties and Actions

Text Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

Track and Spatial Track Properties and Actions are available for Text Track targets.

> ***Note:*** *The target Text Track's Edit State must be set to* **`kScriptEditing`** *using the SetTextEditState action before you can alter the appearance or the data contained in the text track (see SetTextEditState).*

GetEditState — Text Track Properties

Syntax ***GetEditState***

Description This property returns current edit state of the target text track.

The available edit states are:

```
kNoEditing
kDirectEditing
kScriptEditing
```

Return A numeric value indicating the current edit state of the target text track.

Example

QT Version 5.0 or later

See also

GetIdleFrequency — Text Track Properties

Syntax ***GetIdleFrequency***

Description This property returns the idle frequency in movie time in 1/60 second increments.

Return Numeric value indicating the idle frequency.

Example

```
LocalVars idleFreq
idleFreq = GetIdleFrequency
// Get the idle frequency of the current track
```

QT Version 5.0 or later

See also SetIdleFrequency

GetSelectionStart — Text Track Properties

Syntax `GetSelectionStart`

Description This property returns numeric start position of the current selection in the target text track.

Return A numeric value indicating the current selection start position in the target text track.

Example

QT Version 5.0 or later

See also ReplaceText, GetSelectionEnd, GetText

GetSelectionEnd — Text Track Properties

Syntax `GetSelectionEnd`

Description This property returns numeric end position of the current selection in the target text track.

Return A numeric value indicating the current selection end position in the target text track.

Example

QT Version 5.0 or later

See also ReplaceText, GetSelectionStart, GetText

Syntax ***GetText****(start, end)*

start A numeric literal or variable specifying the location of the first character position to retrieve.

end A numeric literal or variable specifying the locationof the last character position to retrieve.

Description This action retrieves the specified selection of characters (as defined by the parameters *start* and *end*).

The *start* and *end* specifies a selection of characters. In order to specify the starting character, you must specify the character position (1 based) of the character preceding the start character of your selection (i.e. Specify 0 to select from the first character). In order to specify the ending character, you must specify the character position (1 based) of the character following the last character in your selection.

Example

```
LocalVars theString, theText
SetString(theString, "This is the first string")
SetString(theText, GetText(0, 5))
// The resulting string now contains "this"
```

QT Version 5.0 or later

See also ReplaceText

GetTextBoxBottom — Text Track Properties

Syntax `GetTextBoxBottom`

Description This property returns the location of the bottom of the text box in the coordinates of the target text track.

Return A numeric value indicating location of the bottom edge of the text box of the target text track.

Example

QT Version 5.0 or later

See also GetTextBoxLeft, GetTextBoxRight, GetTextBoxTop, SetTextBox

GetTextBoxLeft — Text Track Properties

Syntax `GetTextBoxLeft`

Description This property returns the location of the left side of the text box in the coordinates of the target text track.

Return A numeric value indicating location of the left edge of the text box of the target text track.

Example

QT Version 5.0 or later

See also GetTextBoxBottom, GetTextBoxRight, GetTextBoxTop, SetTextBox

GetTextBoxRight — Text Track Properties

Syntax `GetTextBoxRight`

Description This property returns the location of the right side of the text box in the coordinates of the target text track.

Return A numeric value indicating location of the right edge of the text box of the target text track.

Example

QT Version 5.0 or later

See also GetTextBoxBottom, GetTextBoxLeft, GetTextBoxTop, SetTextBox

GetTextBoxTop — Text Track Properties

Syntax `GetTextBoxTop`

Description This property returns the location of the top of the text box in the coordinates of the target text track.

Return A numeric value indicating location of the top edge of the text box of the target text track.

Example

QT Version 5.0 or later

See also GetTextBoxBottom, GetTextBoxLeft, GetTextBoxRight, SetTextBox

Syntax `GetTextLength`

Description This property returns the number of characters in the target text track.

Return A numeric value indicating the number of characters in the target text track. The text length includes spaces.

Example

QT Version 5.0 or later

See also

Text Track Actions

EatKeyEvent — Text Track Actions

Syntax ***EatKeyEvent***

Description This action prevents the text track from processing the Key Pressed event. This action must be called from within the Key Pressed Event handler.

Example

QT Version 5.0 or later

See also

EnterText — Text Track Actions

Syntax ***EnterText***`(character)`

`character` A string literal or variable specifying the character to use.

Description This action replaces the current selection in the target text track with the specified character.

This action is useful for simulating a text entry field using a text track sample. As the characters are typed, the KeyPressed event is triggered so that each character can be sent to the text sample to simulate text entry.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetSelection, GetSelectionStart, GetSelectionEnd

Syntax ***FindText(****string, red, green, blue, flags****)***

string A string literal or variable containing the text to search for.

red A numeric literal specifying the red component of the found text color.

green A numeric literal specifying the green component of the found text color.

blue A numeric literal specifying the blue component of the found text color.

flags A numeric literal or variable specifying the search settings.

Description This action searches and marks the specified text in the target text track. *string* supplies the text to search for while *red*, *green*, *blue* parameters supply the found text color. The movie time will advance to the sample where the text is found if the `kSearchCurrentSample` flag is not used.

The available search flags are:

```
kSearchCurrentSample
kSearchCaseSensitive
kSearchReverse
kSearchWraparound
kSearchAgain
```

The search flags may be used together by adding the constants together (i.e. `kSearchCaseSensitive + kSearchReverse`).

The selection will be set if the specified text was found so that you can retrieve the start and end of the text in the target text track.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

```
SetTextEditState(kScriptEditing)
FindText("LiveStage", 10000, 10000, 10000,
kSearchCaseSensitive + kReverse)
// Search backwards to find the word "LiveStage"
// and mark the text with a gray color
```

QT Version 5.0 or later

See also GetSelectionStart, GetSelectionEnd

HandleMouseClick — Text Track Actions

Syntax ***HandleMouseClick***

Description This action passes the mouse events to the text track allowing it to make text selections and insertion point changes.

This action is effective when the kScriptEditing flag is set using the SetTextEditState action.

Example

QT Version 5.0 or later

See also SetTextEditState, GetEditState

ReplaceText — Text Track Actions

Syntax ***ReplaceText(****string, start, end****)***

string — A string literal or variable containing the text used to replace the specified selection.

start — A numeric literal or variable specifying the location of the first character to replace.

end — A numeric literal or variable specifying the location of the last character to replace.

Description This action replaces the specified selection of characters (as defined by the parameters *start* and *end*) with the characters in the supplied *string*.

Note: *The character positions begin with 0. The end character index you specify must be 1 greater than the character position. Thus in the string "Hello World!", you must specify 0 for start and 5 for end in order to replace the word "Hello".*

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

```
LocalVars theText
SetTextEditState(kScriptEditing)
SetString(theText, "This is the first string")
ReplaceText(theText, 0, 1000)
// Replace the first 1000 characters in the
// Target text track with the value in theText
```

QT Version 5.0 or later

See also GetSelectionStart, GetSelectionEnd, GetText

Syntax ***ScrollText(****xdelta, ydelta****)***

xdelta A numeric literal or variable specifying the amount and direction to scroll the text horizontally.

ydelta A numeric literal or variable specifying the amount and direction to scroll the text vertically.

Description This action scrolls the text in the target text track. To scroll the text in the reverse direction (i.e. scrolling from left to right or scrolling top to bottom), specify a negative scroll value.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

```
SetTextEditState(kScriptEditing)
ScrollText(-10, 0)
// Scroll the text from left to right by 10 pixels
```

QT Version 5.0 or later

See also SetTextScrollDelay

SetBackgroundColor — Text Track Actions

Syntax ***SetBackgroundColor(****red, green, blue****)***

red — A numeric literal specifying the red component of the background color.

green — A numeric literal specifying the green component of the background color.

blue — A numeric literal specifying the blue component of the background color.

Description This action sets the background color of the target text track.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextColor

SetIdleFrequency — Text Track Actions

Syntax ***SetIdleFrequency(****frequency****)***

frequency — An integer expression of the frequency of the idle events in 1/60 second increments.

Description This action will change the idle frequency of the text track. Idle Frequency determines how often the Idle Event handlers are called by QuickTime.

Example

```
SetIdleFrequency(60)
// Set the idle frequency of the track to once
// every second
```

QT Version 5.0 or later

See also

Syntax ***SetSelection(****start, end****)***

start A numeric literal or variable specifying the location of the first character in the selection.

end A numeric literal or variable specifying the location of the last character in the selection.

Description This action selects the specified characters (as indicated by the *start* and *end* parameters) in the target text track.

Note: *The character positions begin with 0. The end character index you specify must be 1 greater than the character position. Thus in the string "Hello World!", you must specify 0 for start and 5 for end in order to select the word "Hello".*

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also GetSelectionStart, GetSelectionEnd, ReplaceText

Syntax ***SetTextAlignment(****alignment****)***

alignment A numeric literal or variable indicating text alignment.

Description This action sets the text alignment for the current text selection in the target text track.

The available text alignments are:

kCenterJustify
kLeftJustify
kRightJustify

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextFont, SetTextStyle, SetTextSize

Syntax ***SetTextBox(****left, top, right, bottom****)***

left A numeric literal or variable specifying the location of the bottom of the left side of the text box.

top A numeric literal or variable specifying the location of the bottom of the top of the text box.

right A numeric literal or variable specifying the location of the bottom of the right side of the text box.

bottom A numeric literal or variable specifying the location of the bottom of the text box.

Description This action sets the text box coordinates of the target text track.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also GetTextBoxLeft, GetTextBoxTop, GetTextBoxRight, GetTextBoxBottom

Syntax	***SetTextColor(****red, green, blue****)***
red	A numeric literal specifying the red component of the foreground color.
green	A numeric literal specifying the green component of the foreground color.
blue	A numeric literal specifying the blue component of the foreground color.
Description	This action sets the foreground color of the current text selection.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example	
QT Version	5.0 or later
See also	SetBackgroundColor

Syntax ***SetTextDisplayFlags(****flags****)***

flags A numeric literal or variable indicating the display flags setting.

Description This action sets the display flags of the target text track.

The available display flags are:

```
kDontDisplay
kDontAutoScale
kClipToTextBox
kUseMovieBGColor
kShrinkTextBoxToFIt
kScrollIn
kScrollOut
kHorizScroll
kReverseScroll
kContinuousScroll
kFlowHoriz
kContinuousKaraoke
kDropShadow
kAntiAlias
kKeyedText
kInverseHilite
kTextColorHilite
```

The above flags can be used together to set multiple flags at the same time.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

```
SetTextEditState(kScriptEditing)
SetTextDisplayFlags(kScrollIn + kScrollOut +
kHorizScroll)
// Sets the flags to scroll in and out horizontally
```

QT Version 5.0 or later

See also

SetTextDropShadow — Text Track Actions

Syntax ***SetTextDropShadow(****xOffset, yOffset, transparency****)***

xOffset A numeric literal or variable indicating the horizontal offset of the dropshadow.

end A numeric literal or variable indicating the vertical offset of the dropshadow.

transparency A numeric literal or variable indicating the degree of transparency of the dropshadow.

Description This action sets the dropshadow parameters used for the target text track.

Note: *This action is valid only if the kDropShadow flag is set for the target text track. (see SetTextDisplayFlags)*

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextHilite, SetTextDisplayFlags

SetTextEditState — Text Track Actions

Syntax ***SetTextEditState(****state****)***

state A numeric literal or variable indicating the text edit state.

Description This action sets the text edit state of the target text track.

The available edit states are:

Edit State	Meaning
kNoEditing	The target text track is not editable by either a script or the user
kDirectEditing	The target text track is editable by the user but NOT by a script
kScriptEditing	The target text track is editable by a script but NOT the user

Example

QT Version 5.0 or later

See also GetEditState, HandleMouseClick

SetTextFont — Text Track Actions

Syntax ***SetTextFont(****fontID****)***

fontID A numeric literal or variable indicating the Font ID.

Description This action sets the font for the current selection in the target text track.

The available font IDs are:

```
kAthensFont
kCairoFont
kCourierFont
kGenevaFont
kHelveticaFont
kLondonFont
kLosAngelesFont
kMobileFont
kMonacoFont
kNewYorkFont
```

```
kSanFranciscoFont
kSymbolFont
kTimesFont
kTorontoFont
kVeniceFont
```

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextStyle, SetTextSize

SetTextHilite — Text Track Actions

Syntax ***SetTextHilite****(start, end, red, green blue)*

start A numeric literal or variable indicating the start position of the selection.

end A numeric literal or variable indicating the end position of the selection.

red A numeric literal indicating the red component of the hilite color.

green A numeric literal indicating the green component of the hilite color.

blue A numeric literal indicating the blue component of the hilite color.

Description This action hilites the specified selection of characters (as defined by the start and end parameters) in the target text track using the supplied color.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextDropShadow

SetTextLinkColor — Text Track Actions

Syntax ***SetTextLinkColor(****index, red, green, blue****)***

index A numeric literal or variable indicating the index number of the hyper link.

red A numeric literal specifying the red component of the text link color.

green A numeric literal specifying the green component of the text link color.

blue A numeric literal specifying the blue component of the text link color.

Description This action sets the specified hyper link color in the target text track.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextLinkStyle

Syntax ***SetTextLinkStyle(****index, style****)***

index A numeric literal or variable indicating the index number of the hyperlink.

style A numeric literal or variable indicating the Font Style.

Description This action sets the font style for the specified (*index*) hyper link text in the target text track.

The available text styles are:

kNormalFace
kBoldFace
kItalicFace
kUnderLineFace
kOutlineFace
kShadowFace
kCondenseFace
kExtendFace

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextLinkColor

SetTextScrollDelay — Text Track Actions

Syntax ***SetTextScrollDelay(****delay****)***

delay A numeric literal or variable indicating amount of time between scrolling in and scrolling out.

Description This action sets the amount of time that is spent between scrolling in and scrolling out in the target text track. This action works with the `kScrollIn` and `kScrollOut` flags used with the SetTextDisplayFlags action.

When used with the `kScrollIn` or the `kScrollOut` display flags, this action sets the amount of time the text is displayed between the Scroll-in and Scroll-out transitions.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also ScrollText

SetTextSize — Text Track Actions

Syntax ***SetTextSize(****size****)***

size A numeric literal or variable indicating the Font Point Size.

Description This action sets the font size in points for the current selection in the target text track.

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextFont, SetTextStyle

Syntax ***SetTextStyle(****style****)***

style A numeric literal or variable indicating the Font Style.

Description This action sets the font style for the current selection in the target text track.

The available text styles are:

```
kNormalFace
kBoldFace
kItalicFace
kUnderLineFace
kOutlineFace
kShadowFace
kCondenseFace
kExtendFace
```

Note: *In order to use this action, you must first set the edit mode of the Text Track to* ***kScriptEditing*** *(See SetTextEditState).*

Example

QT Version 5.0 or later

See also SetTextFont

VR Track Properties and Actions

VR Track properties and actions need to have a track and movie target specified. If no movie target is specified then the current movie is considered to be the target. You can specify a movie target by using MovieNamed(name) or MovieOfID(id) and pass in either the name of the movie or its ID. If no track target is specified then the track that contains the object that is currently executing the script is considered to be the target. You can specify a track target by using TrackNamed(name), TrackOfID(id), TrackOfIndex(index) or TrackOfType(type). See the section on targets for a complete description and some examples of how to specify a target.

VR Tracks are also Spatial Tracks as they have a visual representation within the movie. Thus the Spatial Track Actions and Properties are also available for VR Track Targets (see the section on Targets).

VR Track Properties

GetIdleFrequency	**Sprite Track Properties**
Syntax	***GetIdleFrequency***
Description	This property returns the idle frequency in movie time in 1/60 second increments.
Return	Numeric value indicating the idle frequency.

Example	`LocalVars idleFreq`
	`idleFreq = GetIdleFrequency`
	`// Get the current track's idle frequency`

QT Version	5.0 or later
See also	SetIdleFrequency

ViewHorizCenter — VR Track Properties

Syntax ***ViewHorizCenter***

Description This property returns the horizontal view center.

Return A numeric value indicating the horizontal view center.

Example

QT Version 5.0 or later

See also ViewVertCenter

ViewVertCenter — VR Track Properties

Syntax ***ViewVertCenter***

Description This property returns the vertical view center.

Return A numeric value indicating the vertical view center.

Example

QT Version 5.0 or later

See also ViewHorizCenter

AreHotSpotsVisible — VR Track Properties

Syntax ***AreHotSpotsVisible***

Description This property returns TRUE if the HotSpots are shown.

Return A boolean value indicating TRUE if the HotSpots are visible.

Example

```
if (TrackNamed("MyPano").AreHotSpotsVisible = TRUE)
// Script executes only when hotspots are visible
EndIf
```

QT Version 5.0 or later

See also ShowHotSpots, HideHotSpots

PanAngle — VR Track Properties

Syntax ***PanAngle***

Description This property returns the side to side (Pan) angle of the VR track. The value returned is between 0 and 360 degrees. The pan angle of a VR track indicates the direction the track is pointing.

Return Numeric value between 0 and 360 degrees indicating the pan angle.

Example

```
If (TrackOfID(3).PanAngle = 0)
// Script executes only when PanAngle is 0
EndIf
```

QT Version 3.0 or later

See also SetPanAngleTo, SetPanAngleBy

TiltAngle — VR Track Properties

Syntax ***TiltAngle***

Description This property returns the up and down (Tilt) angle of the VR track. The value returned is -90 to 90 degrees. A value of 90 indicates that the VR track is tilted vertically upwards while a value of -90 indicates that the track is tilted vertically downwards. A value of 0 indicates the tilt angle is set to the horizontal (no tilt).

Return Numeric value between -90 and 90 degrees.

Example

```
If (TrackOfID(3).TiltAngle = 0)
// Script executes only when TiltAngle is 0
EndIf
```

QT Version 3.0 or later

See also SetTileAngleTo, SetTiltAngleBy

FieldOfView — VR Track Properties

Syntax ***FieldOfView***

Description This property returns the Field of View setting of the VR track. Changing the Field of View changes the zoom factor of the VR track.

Return Numeric value indicating the Field of View setting of the VR track.

Example The following example zooms in by x2:

```
TrackNamed("VR
Movie").SetFieldOfViewTo(TrackNamed("VR
Movie").FieldOfView / 2)
```

The following example zooms out by x2:

```
TrackNamed("VR
Movie").SetFieldOfViewTo(TrackNamed("VR
Movie").FieldOfView * 2)
```

QT Version 3.0 or later

See also SetFieldOfViewTo, SetFieldOfViewBy

NodeID — VR Track Properties

Syntax ***NodeID***

Description This property returns the ID of the current Node in the VR track. A node in a VR track contains the necessary data for displaying a VR panorama. A unique ID identifies each node in a VR track.

Return Numeric value indicating the ID of the current Node in the VR track.

Example

```
If (TrackOfID(3).NodeID = 21) ...
```

QT Version 3.0 or later

See also GotoNodeID

VR Track Actions

EnableHotSpot — VR Track Actions

Syntax	***EnableHotSpot****(HotSpot_ID, enable)*
HotSpot_ID	A numeric literal, expression or variable indicating the ID of the VR hotspot to enable/disable.
enable	A boolean literal, expression or variable indicating that the hotspot is to be enabled or disabled
Description	This action enables or disables the specified VR HotSpot.
Example	`ThisTrack.EnableHotSpot(101, FALSE)` `// Disable the hotspot with the ID of 101`
QT Version	5.0 or later
See also	DisableHotspot

GoToNodeID — VR Track Actions

Syntax	***GoToNodeID****(Node_ID)*
Node_ID	An integer expression indicating the ID of the VR node to go to.
Description	This action sends a VR track to the specified Node. The node ID of 2147483648 (hex 80000000) will go to the previous node and the node ID of 2147483649 (hex 80000001) will go to the default node.
Example	`TrackNamed("Room"). GoToNodeID(100)`
QT Version	3.0 or later
See also	Numeric Expression, NodeID

HideHotSpots — VR Track Actions

Syntax ***HideHotSpots***

Description This action hides the VR HotSpots for the current VR node.

Example

```
ThisTrack.HideHotSpots
// Hide the hotspot
```

QT Version 5.0 or later

See also AreHotSpotsVisible, ShowHotSpots

SetFieldOfViewBy — VR Track Actions

Syntax ***SetFieldOfViewBy(****Angle****) <MIN(****number****) MAX(****number****) wraparound>***

Angle A numeric expression indicating the change in the field of view angle.

Description This action changes the Field of View of a VR track by the specified amount. See the description in SetFieldOfViewTo for an explanation of what the field of view is.

Example

```
TrackNamed("Room").SetFieldOfViewBy(5)
```

QT Version 3.0 or later

See also Numeric Expression SetFieldOfViewTo, FieldOfView

SetFieldOfViewTo — VR Track Actions

Syntax ***SetFieldOfViewTo(Angle) <MIN(****number****) MAX(****number****)>***

Angle A numeric expression indicating the field of view angle.

Description This action sets the Field of View of a VR track. The field of view is the angle formed by a line connecting the bottom of the image, to the view point, to the top of the image. A

larger field of view will distort the image but will allow you to see more of it at once. The pan and tilt angles may be adjusted to keep the image in view.

Example `TrackNamed("Room").SetFieldOfViewTo(90)`

QT Version 3.0 or later

See also Numeric Expression, FieldOfView, SetFieldOfViewBy

SetIdleFrequency — Sprite Track Actions

Syntax ***SetIdleFrequency(****frequency****)***

frequency An integer expression setting the new frequency of the idle events in 1/60 second increments.

Description This action will change the idle frequency of the sprite track. Idle Frequency determines how often the Idle Event handlers are called by QuickTime.

Example

```
SetIdleFrequency(1 * 60)
// Set the idle frequency of the track to once
// every second
```

QT Version 5.0 or later

See also

SetPanAngleBy — VR Track Actions

Syntax ***SetPanAngleBy(****Angle****)*** ***<MIN(****number****) MAX(****number****) wraparound>***

Angle — A numeric expression indicating the change in the pan angle.

Description This action modifies the pan angle of a VR track. The parameter Angle specifies the number of degrees to modify the side to side angle of the VR track being displayed.

Example The following statement causes the VR track named Room to pan continuously (if called from the Idle Handler) to the right by 5 degrees wrapping around when the angle reaches 360:

```
SetPanAngleBy(5) MIN(0) MAX(359) wraparound
```

QT Version 3.0 or later

See also Numeric Expression, SetPanAngleTo, PanAngle

SetPanAngleTo — VR Track Actions

Syntax ***SetPanAngleTo(****Angle****)*** ***<MIN(****number****) MAX(****number****)>***

Angle — A numeric expression indicating the new pan angle.

Description This action sets the pan angle of a VR track. The parameter Angle specifies the side to side angle of the VR track to display.

Example The following statement sets the pan angle of the VR track named Room to 0:

```
TrackNamed("Room").SetPanAngleTo(0)
```

QT Version 3.0 or later

See also Numeric Expression, SetPanAngleBy, PanAngle

SetTiltAngleBy — VR Track Actions

Syntax ***SetTiltAngleBy*** *(Angle)* ***<MIN(****number****) MAX(****number****) wraparound>***

Angle — A numeric expression indicating the change in the tilt angle.

Description This action modifies the tilt angle of a VR track. The parameter Angle specifies the number of degrees to modify the up and down angle of the VR track being displayed.

Example The following statement causes the VR track named Room to tilt upwards continuously (if added to the Idle Handler) by 5 degrees wrapping around when the angle reaches 90:

```
SetTiltAngleBy(5) MIN(-90) MAX(90) wraparound
```

QT Version 3.0 or later

See also Numeric Expression, TiltAngle, SetTiltAngleTo

SetTiltAngleTo — VR Track Actions

Syntax ***SetTiltAngleTo*** *(Angle)* ***<MIN(****number****) MAX(****number****)>***

Angle A numeric expression indicating the new tilt angle.

Description This action sets the tilt angle of a VR track. The parameter Angle specifies the up and down angle of the VR track to display.

Example The following statement causes the VR track named Room to reset the tilt angle to 0 (horizontal):

```
TrackNamed("Room").SetTiltAngleTo(0)
```

QT Version 3.0 or later

See also Numeric Expression, SetTiltAngleBy, TiltAngle

ShowDefaultView — VR Track Actions

Syntax ***ShowDefaultView***

Description This action returns a VR track to its default view.

Example The following statement causes the VR track named "Room" to return to its default view:

```
TrackNamed("Room").ShowDefaultView
```

QT Version 3.0 or later

See also SetFieldOfViewTo SetPanAngleTo SetTiltTo

ShowHotSpots — VR Track Actions

Syntax ***ShowHotSpots***

Description This action shows the VR HotSpots for the current VR node.

Example

```
ThisTrack.ShowHotSpots
// Show the hotspots
```

QT Version 5.0 or later

See also AreHotSpotsVisible, HideHotSpots

Index

F

G

L

M

N

P

Q

S

T

V