JavaScript Scripting Guide for QuickTime

2005-10-04

Contents

Tables and Listings

Introduction to JavaScript Scripting Guide for QuickTime

JavaScript can interact with a QuickTime browser plug-in in several ways. You can use JavaScript in a browser to detect whether QuickTime is installed. You can also make QuickTime movies that issue JavaScript commands to the browser, optionally making use of functions defined in a surrounding web page. And for browsers that support two-way communication between JavaScript and plug-ins, you can use JavaScript to query and control the QuickTime plug-in directly.

This document describes client-side scripting using JavaScript and QuickTime browser plug-ins. All QuickTime browser plug-ins expose the exact same interfaces to JavaScript, whether the plug-in is an ActiveX control, a Cocoa plug-in, or a Netscape-style plug-in, allowing the same script to operate identically for Internet Explorer for Windows, Safari for the Mac OS, FireFox, Mozilla, and other browsers.

Who Should Read This Document

If you create QuickTime movies that are embedded in web pages, or if you create web pages that include QuickTime movies, you should read this document.

Organization of This Document

This document provides a detailed description of the different ways to use JavaScript for client-side scripting of QuickTime browser plug-ins:

- Using JavaScript to Detect QuickTime
- Executing JavaScript Functions From QuickTime
- Controlling QuickTime Using JavaScript
- QuickTime JavaScript Reference

See Also

If you want to do Windows application development or server-side scripting using the QuickTime ActiveX control and JavaScript, Visual Basic, or C#, see *QuickTime 7 for Windows Update Guide*.

For information on how to embed QuickTime content in a web page, or how to control the QuickTime browser plug-in using HTML tags and attributes, see *HTML Scripting Guide for QuickTime*.

For information on how to control QuickTime and the QuickTime browser plug-in using SMIL, see *SMIL Scripting Guide for QuickTime*.

QuickTime and JavaScript

Recent versions of the QuickTime plug-in are fully scriptable using JavaScript in most browsers for the Mac OS and Windows. The degree of interactivity varies significantly by QuickTime version, operating system, and browser.

There are three levels of JavaScript interactivity supported by QuickTime: you can use JavaScript to detect whether QuickTime is installed; you can issue JavaScript commands and call JavaScript functions from a QuickTime movie; and you can control the QuickTime plug-in directly using JavaScript.

The QuickTime browser plug-in comes in multiple flavors. There are traditional Netscape-style plug-ins for the Mac OS and Windows, COM objects and ActiveX controls for Internet Explorer on Windows, and a Cocoa plug-in for Safari. All of these plug-ins present the exact same interfaces to JavaScript.

In addition to the QuickTime browser plug-in, QuickTime 7 and later include an ActiveX control that is scriptable using the COM interface from Visual Basic, JavaScript, or C#. That ActiveX control is not the same as the ActiveX version of the QuickTime browser plug-in. This document describes the JavaScript interface to QuickTime browser plug-ins, for client-side scripts running in a browser. If you want to develop Windows desktop applications or server-side scripts on the Windows OS using JavaScript and QuickTime, see *QuickTime 7 for Windows Update Guide*.

Using JavaScript to Detect QuickTime

JavaScript can be used to detect the QuickTime plug-in in most versions of most browsers on Windows and the Mac OS. The details vary slightly by browser, but a single script can be written that works with all supported browsers.

Most browsers other than Internet Explorer for Windows (including Netscape, Mozilla-based browsers, Safari, and Internet Explorer for Macintosh version 5) support the JavaScript `navigator.plugins.name` array. You can detect whether the user has the QuickTime plug-in installed by testing this array for “QuickTime.”

Internet Explorer for Windows does not support the `navigator.plugins` array, but does allow you to use VBScript to detect whether the QuickTime plug-in is installed (by testing for the presence of the QuickTime COM object).

Note: The QuickTime COM object was introduced as part of QuickTime 4.1

It is not possible to detect the presence of the QuickTime plug-in using JavaScript in versions of Internet Explorer for Macintosh prior to version 5. Since QuickTime is an integral part of the Mac OS, however, it is safe to assume that users running older versions of Internet Explorer on the Mac OS do have QuickTime installed.

The following listing contains an example script that detects the user's operating system, browser type, and browser version, uses JavaScript to test for the QuickTime plug-in, uses VBScript to test for the QuickTime COM object, and detects users running older versions of Internet Explorer for Macintosh (making the assumption that these users also have QuickTime installed). This listing sets the variable `haveqt` either `true` or `false`.

Listing 1-1 Detecting QuickTime with JavaScript

```
<HEAD>
<TITLE>Test for QuickTime</TITLE>

<SCRIPT LANGUAGE="Javascript">
var haveqt = false;
</SCRIPT>

<SCRIPT LANGUAGE="VBScript">
On Error Resume Next
Set theObject =
CreateObject("QuickTimeCheckObject.QuickTimeCheck.1")
On Error goto 0
If IsObject(theObject) Then
If theObject.IsQuickTimeAvailable(0) Then
haveqt = true
End If
End If
</SCRIPT>

<SCRIPT LANGUAGE="Javascript">

if (navigator.plugins) {
for (i=0; i < navigator.plugins.length; i++ ) {
if (navigator.plugins[i].name.indexOf
("QuickTime") >= 0)
{ haveqt = true; }
}
}

if ((navigator.appVersion.indexOf("Mac") > 0)
&& (navigator.appName.substring(0,9) == "Microsoft")
&& (parseInt(navigator.appVersion) < 5) )
{ haveqt = true; }

</SCRIPT>
</HEAD>
```

If you insert this JavaScript into the `<HEAD>` element of your HTML, you can test the variable `haveqt` for the presence of QuickTime. You might, for example, want to write the `<EMBED>` and `<OBJECT>` tags for a movie using JavaScript, writing the tags only if QuickTime is installed. Or you might want to redirect users without QuickTime to an alternate page.

Executing JavaScript Functions From QuickTime

Any time QuickTime sends a URL to the browser, that URL can be a short set of in-line JavaScript commands or a call to a JavaScript function declared on the current HTML page, including any necessary parameters.

Several things can cause QuickTime to send a URL to the browser, such as the user clicking a hotspot in a QuickTimeVR panorama, clicking a movie that has an associated HREF, loading a sample in an HREF track (a type of text track containing URLs), or as the result of a wired action. Wired actions can be triggered by user interaction, as a result of a frame in the movie being displayed, or as the result of arbitrary wired calculations.

Sending URLs to a browser is supported in QuickTime 3 and later for movie HREFs, visual track HREFs, HREF track HREFs, and VR hotspots. Wired sprite actions in general are supported in QuickTime 4 and later (new wired actions have been introduced with various releases of QuickTime, so a particular wired action may require a later QuickTime version).

Sending JavaScript to the browser as part of a QuickTime URL is supported on all browsers that support JavaScript on Windows or the Mac OS, which includes all recent versions of all common browsers.

The following listing contains an example of an HTML page that causes QuickTime to call to a JavaScript function or send a series of in-line JavaScript commands in response to a mouse click in a QuickTime movie.

Listing 1-2 Calling JavaScript from QuickTime

```
<HTML>
<HEAD>
<TITLE>Calling JavaScript from QuickTime</TITLE>

<script language="JavaScript">
<!--
function openQTwin(url) {
qtwin = window.open(url,"moviewindow",'toolbar=0,location=0,
directories=0,status=0,menubar=0,scrollbars=0,resizable=0, width=360,height=260');
qtwin.focus();
}
// -->
</script>

</HEAD>

<BODY align=center>

<P>Calling a JavaScript function: <BR>
<EMBED HEIGHT=320 WIDTH=240
```

```
SRC="Poster.mov" TYPE="video/quicktime"
CONTROLLER="false"
HREF="javascript:openQTwin('MyMovie.mov')" />

<P>
Click the poster above to play a MyMovie in a new window.
</P>

<P>Passing in-line JavaScript commands: <BR>
<EMBED HEIGHT=320 WIDTH=240
SRC="Poster2.mov" TYPE="video/quicktime"
CONTROLLER="false"
HREF="javascript: alert('You clicked the poster!')" />

<P>
Click the poster above to show an alert box.
</P>

</BODY>
</HTML>
```

Controlling QuickTime Using JavaScript

QuickTime exposes a powerful set of objects, properties, and methods to JavaScript. The objects include QuickTime itself, the browser plug-in, any embedded movies, and all the tracks within those movies.

The methods allow you to control a movie—start it, stop it, step it forward or back, or replace it with another movie. Setting properties allows you to control how a movie behaves—enable and disable tracks, select a language, change a video track's size, position, and rotation, modify a sound track's volume, set the movie's rate of play and direction, set or unset looping, and so on.

Getting properties allows you to obtain information—the installed version of the QuickTime plug-in, the duration of a movie, what percentage of it has been downloaded, whether it is playing or has finished, how many tracks it has, and more.

Recent versions of the QuickTime plug-in are fully scriptable using JavaScript in most browsers for the Mac OS and Windows. There are exceptions, however, depending on the QuickTime version, browser type, browser version, and operating system.

Supported Interfaces

Not all browsers support communication between JavaScript and plug-ins. Those that support such communication do so using a variety of interfaces, including LiveConnect, COM (ActiveX), XPCOM, and Cocoa. QuickTime currently supports all of these interfaces.

Note: Netscape first introduced JavaScript support using the LiveConnect interface. Netscape 6, FireFox, and Mozilla 1.0 support JavaScript using the newer XPCOM interface. Safari 1.3 and later also support a Cocoa interface. Internet Explorer for Windows allows JavaScript to interact with plug-ins using the COM interface.

Support for LiveConnect was added to QuickTime in version 4.1. Support for COM, XPCOM, and Cocoa were added in QuickTime 6.

Supported Browsers and Operating Systems

The QuickTime plug-in is scriptable from all browsers that support the LiveConnect, XPCOM, or COM interface. This includes all versions of Netscape and Mozilla for Windows and Macintosh, AOL 5 and later for Windows, MSN 6 and later, and all versions of Internet Explorer for Windows.

In QuickTime 7 and later, the QuickTime plug-in is scriptable from the Safari browser; Safari 2.0 or later is required (Safari 1.3 or later is supported on Panther).

Internet Explorer for Macintosh does not allow scripting of any plug-in.

The Mozilla Foundation, in collaboration with Apple, Adobe, Macromedia, Opera, and Sun Microsystems, are currently working together to define a new NPAPI interface to allow greater interactivity between standards-based browsers and plug-ins. Check the Apple website for updates concerning QuickTime and NPAPI-compliant browsers.

Before You Start

The browser must load a copy of the QuickTime plug-in before you can query or control QuickTime using JavaScript. In addition, the interface between the browser and the plug-in must be initialized. In most cases, you also want to give an embedded movie a name so it can be addressed by name in your script.

This is accomplished by the following steps:

- Use the HTML `<EMBED>` tag and `<OBJECT>` tag to cause the browser to load a copy of the QuickTime plug-in.
- Set the attribute EnableJavaSript="true" in the `<EMBED>` tag.
- Set the <OBJECT> `id` attribute and the `<EMBED>NAME` attribute to a name for the movie. Use the same name for both attributes.

An example that loads the QuickTime plug-in, enables JavaScript, and gives a name to a movie is shown in listing 1-3.

Listing 1-3 Load QuickTime, enable JavaScript, assign a Nname

```
<OBJECT
classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
codebase="http://www.apple.com/qtactivex/qtplugin.cab"
width="320" height="256"
```

```
id="movie1">

<PARAM name="src" value="MyMovie.mov">

<EMBED HEIGHT=256 WIDTH=320
SRC="MyMovie.mov"
TYPE="video/quicktime"
PLUGINSPAGE="www.apple.com/quicktime/download"
EnableJavaScript="true" /
NAME="movie1"
/>

</OBJECT>
```

See *HTML Scripting Guide for QuickTime*for more information about the `<EMBED>` and `<OBJECT>` tags and the attributes or parameters that can be passed to QuickTime.

Addressing QuickTime Movies

JavaScript treats each embedded QuickTime movie in a web page as a separately addressable object.

All methods are addressed to a movie. Methods that act on a movie are addressed to the target movie. Methods that operate on a track are addressed to the track's parent movie (the track is specified in a parameter). Methods that operate on QuickTime or the QuickTime plug-in can be addressed to any movie embedded in the document.

Movies can be identified by name if there is a `NAME` attribute in the movie's `EMBED` tag and an `id` attribute in the movie's `OBJECT` tag. Internet Explorer for Windows uses the `id` attribute. All other browsers use the `NAME` parameter. Both `NAME` and `id` should be set to the same value.

Because Internet Explorer and some other browsers do not always support the `embeds[]` array, it is recommended that you assign a name to each movie and address the movie by name in your script, instead of addressing movies by their place in the `document.embeds[]` array.

JavaScript Usage Example

Listing 1-4 gives an example of JavaScript usage with QuickTime.

Listing 1-4 Using JavaScript to play, stop, and replace a QuickTime movie

```
<html>
 <head>
        <title>Simple QuickTime Movie Controls</title>
 </head>

 <script language ="JavaScript">

 <!--
        /* define function that calls QuickTime's "Play" method */
        function PlayIt(anObj)
        {
                anObj.Play();
        }
        /* define function that calls QuickTime's "Stop" method */
        function StopIt(anObj)
        {
                anObj.Stop();
        }
 //-->

 </script>
     <body >

                <P>
                This page uses JavaScript to control a QuickTime movie...
                </P>
<div align=center>
      <table>
           <tr>
           <td width=200>
               <OBJECT classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
                       codebase="http://www.apple.com/qtactivex/qtplugin.cab"
                       width="180" height="160"
                       id="movie1" >

                <PARAM name="src" value="My.mov">

                <EMBED width="180" height="160"
                       src="My.mov"
                       TYPE="video/quicktime"
                       PLUGINSPAGE="www.apple.com/quicktime/download"
                       name="movie1"
                       enablejavascript="true">
                </EMBED>
               </OBJECT>
<P> Movie1 </P>

           </td>

           <td width=200>
               <OBJECT classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
                       codebase="http://www.apple.com/qtactivex/qtplugin.cab"
                       width="180" height="160"
                       id="movie2" >

                <PARAM name="src" value="MyOther.mov">

                <EMBED width="180" height="160"
                       src="MyOther.mov"
                       TYPE="video/quicktime"
                       PLUGINSPAGE="www.apple.com/quicktime/download"
                       name="movie2"
                       enablejavascript="true">
                </EMBED>
               </OBJECT>
<P> Movie2 </P>
           </td>
           </tr>
      </table>
</div>
```

```
 <P>Pass movie name as a parameter to JavaScript functions defined locally: <br>
 <a href="javascript:PlayIt(document.movie1);">PlayIt(movie1)</a><br>
 <a href="javascript:StopIt(document.movie1);">StopIt(movie1)</a><br>
 <a href="javascript:PlayIt(document.movie2);">PlayIt(movie2)</a><br>
 <a href="javascript:StopIt(document.movie2);">StopIt(movie2)</a><br>
 </P>

 <P>Control movie by name directly: <br>
 <a href="javascript:document.movie1.Play();">movie1.Play()</a><br>
 <a href="javascript:document.movie1.Stop();">movie1.Stop()</a><br>
 <a href="javascript:document.movie2.Play();">movie2.Play()</a><br>
 <a href="javascript:document.movie2.Stop();">movie2.Stop()</a><br>
 </P>

 <P>Replace a movie by name directly: <br>
 <a
href="javascript:document.movie1.SetURL('MyOther.mov');">movie1.SetURL(MyOther.mov)</a><br>
  <a
href="javascript:document.movie1.SetURL('My.mov');">movie1.SetURL(My.mov)</a><br>
  <a
href="javascript:document.movie2.SetURL('MyOther.mov');">movie2.SetURL(MyOther.mov)</a><br>
  <a
href="javascript:document.movie2.SetURL('My.mov');">movie2.SetURL(My.mov)</a><br>
 </P>

         </body>
 </html>
```

QuickTime JavaScript Reference

QuickTime exposes a number of methods to JavaScript. Some take the form of commands that operate on movies, such as `Play()`, `Stop()`, `Rewind()`, and so on. The rest operate on properties of QuickTime as a whole, the QuickTime plug-in, movies, or tracks. Most of these properties can be read or written using complementary Get and Set methods. Other properties, such as the QuickTime version, are read-only.

Movie Commands

Movie commands are addressed to a specific movie. For example, the following line of code plays a movie whose `NAME` and `id` are set to “Movie1.”

```
document.movie1.Play();
```

Play

```
void Play()
```

Plays the movie at the default rate, starting from the movie’s current time.

Stop

```
void Stop() Example:
```

```
document.movie1.Stop();
```

Stops the movie without changing the movie's current time.

Rewind

```
void Rewind()
```

Sets the current time to the movie's start time and pauses the movie.

Step

```
void Step(int count)
```

Steps the movie forward or backward the specified number of frames from the point at which the command is received. If the movie's rate is non-zero, it is paused.

ShowDefaultView

```
void ShowDefaultView()
```

Displays a QuickTime VR movie's default node, using the default pan angle, tilt angle, and field of view as set by the movie's author.

GoPreviousNode

```
void GoPreviousNode()
```

Returns to the previous node in a QuickTime VR movie (equivalent to clicking the Back button on the VR movie controller).

QuickTime Properties

Methods that get or set a QuickTime property can be addressed to any movie embedded in the document. For example, both lines in the following code snippet return the same information from a page with two embedded movies, named MovieOne and MovieTwo.

```
myVar1=document.MovieOne.GetQuickTimeVersion();
myVar2=document.MovieTwo.GetQuickTimeVersion();
```

GetQuickTimeVersion

```
string GetQuickTimeVersion()
```

Returns the version of QuickTime.

GetQuickTimeLanguage

```
string GetQuickTimeLanguage()
```

Returns the user's QuickTime language (set through the plug-in's Set Language dialog).

GetQuickTimeConnectionSpeed

```
int GetQuickTimeConnectionSpeed()
```

Returns the connection speed setting from the users QuickTime preferences.

GetIsQuickTimeRegistered

```
boolean GetIsQuickTimeRegistered()
```

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

GetComponentVersion

```
string GetComponentVersion(string type, string subType, string manufacturer)
```

Returns the version of a specific QuickTime component. The component is specified using a four character string for the type, subtype, and manufacturer. For example, to check the version of Apple's JPEG graphics importer call `GetComponentVersion>` `'grip','JPEG','appl').'0'` is a wildcard for any field. If the component is not available, `0.0` is returned.

Plug-in Properties

GetPluginVersion

```
string GetPluginVersion()
```

Returns the version of the QuickTime plug-in.

ResetPropertiesOnReload

```
boolean GetResetPropertiesOnReload()
```

```
void SetResetPropertiesOnReload(boolean reset)
```

By default, most movie and plug-in properies are reset when a new movie is loaded. For example, when a new movie loads, the default controller setting is `true` for a linear movie and `false` for a VR movie, regardless of the prior setting. If this property is set to `false`, the new movie inherits the settings in use with the current movie.

Movie Properties

Like movie commands, methods that get or set movie properties are addressed to a specific movie. For example, the following code snippet set the movie named MovieOne to autoplay.

```
document.MovieOne.SetAutoPlay(true);
```

GetPluginStatus

```
string GetPluginStatus()
```

`GetPluginStatus` returns a string with the status of the current movie. Possible states are:

"Waiting"—waiting for the movie data stream to begin
"Loading"—data stream has begun, not able to play/display the movie yet
"Playable"—movie is playable, although not all data has been downloaded
"Complete"—all data has been downloaded
"Error: <error number>"—the movie failed with the specified error number

Note: Even though the method is named `GetPluginStatus` it gets the status of a specific movie, not the status of the plug-in as a whole. If more than one movie is embedded in a document, there can be a different status for each movie. For example, one movie could be playable while another is still loading.

AutoPlay

```
boolean GetAutoPlay()
```

```
void SetAutoPlay(boolean autoPlay)
```

Get and set whether a movie automatically starts playing as soon as it can. The Set method is roughly equivalent to setting the `AUTOPLAY` parameter in the `<EMBED>` tag, but the @HH:MM:SS:FF feature is not yet supported in JavaScript.

ControllerVisible

```
boolean GetControllerVisible()
```

```
void SetControllerVisible(boolean visible)
```

Get and set whether a movie has a visible controller. The Set method is equivalent to setting the `CONTROLLER` parameter in the `<EMBED>` tag.

Rate

```
float GetRate()
```

```
void SetRate(float rate)
```

Get and set the playback rate of the movie. A rate of 1 is the normal playback rate. A paused movie has a rate of 0. Fractional values are slow motion, and values greater than one are fast-forward. Negative values indicate that the movie is playing backward. Setting the rate of a paused movie to a nonzero value starts the movie playing.

Note: Rate goes to zero when the movie finishes playing or is stopped (by the user, for example). You can use the `GetTime` and `GetDuration` functions to determine whether the movie is stopped at the end, the beginning (time zero), or at some point in between. You may also want the call `GetIsLooping` to determine whether the movie will end spontaneously.

Time

```
int GetTime()
```

```
void SetTime(int time)
```

Get and set the current time of a movie. Setting this property causes a movie to go to that time in the movie and stop.

Volume

```
int GetVolume()
```

```
void SetVolume(int volume)
```

Get and set the audio volume of the movie. A negative value mutes the movie. The Set method is equivalent to setting the `VOLUME` parameter in the `<EMBED>` tag.

Mute

```
boolean GetMute()
```

```
void SetMute(boolean mute)
```

Get and set the audio mute of a movie while maintaining the magnitude of the volume, so turning mute off restores the volume.

MovieName

```
string GetMovieName()
```

```
void SetMovieName(string movieName)
```

Get and set a name that can be used by a wired sprite when targeting an external movie. The Set method is equivalent to setting the `MOVIENAME` parameter in the `<EMBED>` tag.

MovieID

```
int GetMovieID()
```

```
void SetMovieID(int movieID)
```

Get and set an ID that can be used by a wired sprite when targeting an external movie. The Set method is equivalent to setting the `MOVIEID` parameter in the `<EMBED>` tag.

Note: `MovieID` is not the same as the `NAME` parameter in the `<EMBED>` tag or the `id` parameter in the `<OBJECT>` tag. `MovieID` is used for wired sprite addressing, not JavaScript addressing.

StartTime

```
void SetStartTime(int time)
```

Get and set the time at which a movie begins to play and the time at which it stops or loops when playing in reverse. Initially, the start time of a movie is set to `0` unless specified in the `STARTTIME` parameter in the `<EMBED>` tag. The start time cannot be set to a time greater than the end time. The Set method is equivalent to setting the `STARTTIME` parameter in the `<EMBED>` tag.

EndTime

```
int GetEndTime()
```

```
void SetEndTime(int time)
```

Get and set the time at which a movie stops playing or loops. The end time of a movie is initially set to its duration, unless specified in the `ENDTIME` parameter in the `<EMBED>` tag. The end time cannot be set to a time greater than the movie's duration. The Set method is equivalent to setting the `ENDTIME` parameter in the `<EMBED>` tag.

BgColor

```
string GetBgColor()
```

`void SetBgColor(string color)`

Get and set the color used to fill any space allotted to the plug-in by the `<EMBED>` tag and not covered by the movie. The Set method is equivalent to setting the `BGCOLOR` parameter in the `<EMBED>` tag and takes the same values. Regardless of the syntax used to specify the color, `GetBgColor()` always returns the color as a number—for example, if the background color is set to `Navy`,`GetBgColor()` returns `#000080.`

IsLooping

`boolean GetIsLooping()`

`void SetIsLooping(boolean loop)`

Get and set whether a movie loops when it reaches its end. A movie can loop either by restarting when it reaches the end or by playing backward when it reaches the end, then restarting when it reaches the beginning, depending on the `LoopIsPalindrome` value. Using the `SetIsLooping` method is equivalent to setting the `LOOP` parameter to `true` or `false` in the `<EMBED>` tag.

LoopIsPalindrome

`boolean GetLoopIsPalindrome()`

`void SetLoopIsPalindrome(boolean loop)`

Get and set whether a looping movie reverses direction when it loops, alternately playing backward and forward. The loop property must be `true` for this to have any effect. Setting both `IsLooping` and `LoopIsPalindrome` to `true` is equivalent to setting the `LOOP` parameter to `Palindrome` in the `<EMBED>` tag.

PlayEveryFrame

`boolean GetPlayEveryFrame()`

`void SetPlayEveryFrame(boolean playAll)`

Get and set whether QuickTime should play every frame in a movie even if it gets behind (playing in slow motion rather than dropping frames). The sound is muted when `playAll` is set `true.` The Set method is equivalent to setting the `PLAYEVERYFRAME` parameter in the `<EMBED>` tag.

HREF

`string GetHREF()`

`void SetHREF(string url)`

Get and set the URL that is invoked by a mouse click in a movie's display area. The URL can specify a web page, a QuickTime movie, a live streaming session, or be a JavaScript function name. The Set method is equivalent to setting the `HREF` parameter in the `<EMBED>` tag.

Target

`string GetTarget()`

`void SetTarget(string target)`

Get and set the target for a movie's `HREF` parameter. The target can be an existing frame or browser window, a new browser window, `myself` (the QuickTime plug-in), or `quicktimeplayer`. The Set method is equivalent to setting the `TARGET` parameter in the `<EMBED>` tag.

QTNEXTUrl

```
string GetQTNEXTUrl(int index)
```

```
void SetQTNEXTUrl(int index, string url)
```

Get and set the URL and target for a specified item in a sequence. The URL of the first item in the sequence is invoked when the currently selected movie finishes. If the URL specifies a QuickTime movie and the special target `myself`, the next specified URL in the sequence is invoked when that movie finishes, and so on. The Set method is equivalent to setting the `QTNEXT`*n* parameter in the `<EMBED>` tag.

URL

```
string GetURL()
```

Returns a movie's full URL.

```
void SetURL(string url)
```

Replaces a movie with another movie specified by the URL.

KioskMode

```
boolean GetKioskMode()
```

```
void SetKioskMode(boolean kioskMode)
```

Set and get whether kiosk mode is currently set. In kiosk mode, the QuickTime plug-in does not allow the viewer to save a movie to disk. Setting `kioskMode` to `true` is equivalent to setting the `KIOSKMODE` parameter in the `<EMBED>` tag.

GetDuration

```
int GetDuration()
```

Returns the length of the movie (in the movie's time scale units).

GetMaxTimeLoaded

```
int GetMaxTimeLoaded()
```

Returns the amount of the movie that has been downloaded (in the movie's time scale units).

GetTimeScale

```
int GetTimeScale()
```

Returns the time scale of the movie in units per second. For example, if `GetTimeScale()` returns `30`, each movie time scale unit represents 1/30 of a second.

GetMovieSize

```
int GetMovieSize()
```

Returns the size of the movie in bytes.

GetMaxBytesLoaded

```
int GetMaxBytesLoaded()
```

Returns the number of bytes of the movie that have been downloaded.

GetTrackCount

```
int GetTrackCount()
```

Returns the total number of tracks in the movie.

Matrix

```
string GetMatrix()
```

```
void SetMatrix(string matrix)
```

Get and set a movie's transformation matrix. QuickTime uses a 3 x 3 transformation matrix, represented in JavaScript by three lines of three numbers separated by commas:

```
a, b, u
c, d, v
h, k, w
```

You can use a movie's transformation matrix to scale, translate, and rotate the movie image. For details on the transformation matrix, see Movie Internals.

Rectangle

```
string GetRectangle()
```

```
void SetRectangle(string rect)
```

Get and set the location and dimensions of the movie within the embed area.

Note: Normally, the QuickTime plug-in keeps the movie centered within the embed area, even if the embed area changes. Once a movie's location is changed with `SetRect` or `SetMatrix`, the movie's absolute location within the embed area is maintained rather than centering it.

Language

```
string GetLanguage()
```

```
void SetLanguage(string language)
```

Get and set the movie's current language. Setting the language causes any tracks associated with that language to be enabled and tracks associated with other languages to be disabled. If no tracks are associated with the specified language, the movie's language is not changed.

Supported language names:

Albanian
Arabic

Belorussian
Bulgarian
Croatian
Czech
Danish
Dutch
English
Estonian
Faeroese
Farsi
Finnish
Flemish
French
German
Greek
Hebrew
Hindi
Hungarian
Icelandic
Irish
Italian
Japanese
Korean
Latvian
Lithuanian
Maltese
Norwegian
Polish
Portuguese
Romanian
Russian
Saamisk
Serbian
Simplified Chinese
Slovak
Slovenian
Spanish
Swedish
Thai
Traditional Chinese
Turkish
Ukrainian
Urdu
Yiddish

GetMIMEType

```
string GetMIMEType()
```

Returns the movie's MIME type.

GetUserData

```
string GetUserData(string type)
```

Returns the movie user data text with the specified tag. The tag is specified with a four-character string; for example, `'©cpy'` returns a movie's copyright string. The following table contains a list of user data tags.

Table 1-1 User data atrings

String	Data
`'©nam'`	Movie's name
`'©cpy'`	Copyright statement
`'©day'`	Date the movie content was created
`'©dir'`	Name of movie's director
`'©ed1'` to `'© ed9'`	Edit dates and descriptions
`'©fmt'`	Indication of movie format (computer-generated, digitized, and so on)
`'©inf'`	Information about the movie
`'©prd'`	Name of movie's producer
`'©prf'`	Names of performers
`'©req'`	Special hardware and software requirements
`'©src'`	Credits for those who provided movie source content
`'©wrt'`	Name of movie's writer

GetIsVRMovie

```
boolean GetIsVRMovie()
```

Returns `true` if the movie is a QuickTime VR movie, `false` otherwise.

VR Movie Properties

VR movie properties are movie properties that are present only for movies that contain VR panoramas or VR objects. To test for these properties, use the method `GetIsVRMovie()`.

HotspotURL

```
string GetHotspotUrl(int hotspotID)
```

```
void SetHotspotUrl(int hotspotID, string url)
```

Get and set the URL associated with a specified VR movie hot spot. The Set method is equivalent to setting the `HOTSPOT`*n* parameter in the `<EMBED>` tag.

HotspotTarget

```
string GetHotspotTarget(int hotspotID)
```

```
void SetHotspotTarget(int hotspotID, string target)
```

Get and set the target for a specified VR movie hot spot. The Set method is equivalent to setting the `TARGET`*n* parameter in the `<EMBED>` tag.

PanAngle

```
float GetPanAngle()
```

```
void SetPanAngle(float angle)
```

Get and set the QuickTime VR movie's pan angle (in degrees). The Set method is equivalent to setting the `PAN` parameter in the `<EMBED>` tag.

TiltAngle

```
float GetTiltAngle()
```

```
void SetTiltAngle(float angle)
```

Get and set the QuickTime VR movie's tilt angle (in degrees).The Set method is equivalent to setting the `TILT` parameter in the `<EMBED>` tag.

FieldOfView

```
float GetFieldOfView()
```

```
void SetFieldOfView(float fov)
```

Get and set the QuickTime VR movie's field of view (in degrees). Setting a narrower field of view causes the VR to "zoom in." Setting a wider field of view causes the VR to "zoom out." The Set method is equivalent to setting the `FOV` parameter in the `<EMBED>` tag.

GetNodeCount

```
int GetNodeCount()
```

Returns the number of nodes in a QuickTime VR movie.

GetNodeID

```
int GetNodeID()
```

Returns the ID of the current node in a QuickTime VR movie.

SetNodeID

```
void SetNodeID(int id)
```

Sets the current node (by ID) in a QuickTime VR movie (the movie goes to the node with the specified ID).

Track Properties

`Track` properties belong to a specific track within a movie. To get or set a track property, you must know the track's parent movie and the ordinal number of the track within the movie. When you get a movie's properties in QuickTime Player, the tracks are listed in numerical order: the first track listed is track 1, the second is track 2, and so on. For example, the following code snippet disables the third track in a movie named MovieOne.

```
document.MovieOne.SetTrackEnabled(3,false);
```

GetTrackName

```
string GetTrackName(int index)
```

Returns the name of the specified track.

GetTrackType

```
string GetTrackType(int index)
```

Returns the type of the specified track, such as video, sound, text, music, sprite, 3D, VR, streaming, movie, Flash, or tween.

TrackEnabled

```
boolean GetTrackEnabled(int index)
```

```
void SetTrackEnabled(int index, boolean enabled)
```

Get and set the enabled state of a track.

SpriteTrackVariable

```
string GetSpriteTrackVariable(int trackIndex,    int variableIndex)
```

```
void SetSpriteTrackVariable(int trackIndex,    int variableIndex, string value)
```

Get and set the specified sprite track variable value in the specified track.

Note: You can get and set sprite variable values only in sprite tracks that already have defined variables. You cannot use JavaScript to create a new variable or add it to a track.

Document Revision History

This table describes the changes to *JavaScript Scripting Guide for QuickTime*.

Date	Notes
2005-10-04	Updated to include Safari 1.3 for Panther and to reference the new ActiveX control for Windows.
2005-06-04	Expanded the partial list of supported languages from 21 to the full set of 46. Added replacing a movie to the sample code.
2005-04-08	New document describing how to use JavaScript with the QuickTime browser plug-in and ActiveX control.