QuickTime and JavaScript

Recent versions of the QuickTime plug-in are fully scriptable using JavaScript in most browsers for the Mac OS and Windows.

There are a few different ways to use JavaScript with QuickTime: you can use JavaScript to detect whether QuickTime is installed; you can use JavaScript to write the OBJECT and EMBED tags used to display QuickTime content; 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, as described in this document.

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*.

Important: Starting with QuickTime 7.1.5, you can no longer issue javascript:// URLs or call JavaScript functions directly from within a QuickTime movie. This feature was removed from QuickTime for security reasons. There are workarounds, however. See "Executing JavaScript Functions From QuickTime" for examples.

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 ActiveX control).

Note: The QuickTime ActiveX control was introduced as part of QuickTime 4.1

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" type="text/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" type="text/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.

Using JavaScript to Embed QuickTime Content

Apple provides a JavaScript utility to generate the required tags to embed QuickTime content in a web page. Using this utility greatly simplifies the necessary code to embed a movie, as it automatically supplies such information as the QuickTime Class ID, code base, and plugins page. This has two main advantages over typing the tags manually: it requires much less code; and it works seamlessly with Internet Explorer for Windows, versions 6 and later, avoiding the “Click OK to enable ActiveX control” dialog box.

To use the utility, first download the JavaScript file, `AC_QuickTime.js`, from http://developer.apple.com/internet/licensejs.html.
Include it in your web page by inserting the following line of code into your HTML head:

```
<script src="AC_QuickTime.js" language="javascript"
type="text/javascript"> </script>
```

Wherever you want QuickTime content to appear in your web page, place a line of JavaScript that calls the function `QT_WriteOBJECT( )`, passing the URL of the QuickTime content, the width and height of the display area, the preferred ActiveX version (typically left blank) and any optional QuickTime attributes.

Important: The fourth parameter, the preferred ActiveX version, is typically left blank, but it can *not* be omitted. It is a mandatory entry.

A simple example follows.

```
<script language="javascript" type="text/javascript">

    QT_WriteOBJECT('MyMovie.mov' , '320', '240', '');

</script>
```

At run time, this will generate the correct `<OBJECT>` and `<EMBED>` tags and insert them into your web page. This example passes the URL `MyMovie.mov`, the filename of a QuickTime movie in the same directory as the web page. The allocated width is 320 pixels, and the allocated height is 240 pixels. The ActiveX version is left blank, so it defaults to the most recent version.

Optional QuickTime parameters are passed as name/value pairs, that is, the name of the attribute is passed, followed by the value. An example follows.

```
<script language="javascript" type="text/javascript">

    QT_WriteOBJECT('../Movies/MyMovie.mov', '100%', '95%', '',
'AUTOPLAY', 'True', 'SCALE', 'Aspect') ;

</script>
```

In this example, the URL is `../Movies/MyMovie.mov`, the allocated width is 100% of the browser window, the allocated height is 95% of the browser window, the ActiveX version is left blank, the `AUTOPLAY` attribute is set to `True`, and the `SCALE` attribute is set to `Aspect`. The movie will autoplay, scaled to fill almost the entire browser window, while preserving the movie's aspect ratio.

Important: Sometimes you need to pass a parameter only to the `<OBJECT>` tag, or only to the `<EMBED>` tag. For example, if you are using DOM events, you need to address objects by `ID` and the `<EMBED>` object and the `<OBJECT>` object must have unique `ID` values. To set a parameter for only the `<OBJECT>` tag, use the prefix `obj#`. To set a parameter for only the `<EMBED>` tag, use the `emb#` prefix.

The following example passes different IDs to the `<OBJECT>` and `<EMBED>` tags, and sets a name parameter for the `<EMBED>` tag that matches the `<OBJECT>` id. This allows you to monitor DOM events using the ID, regardless of whether the browser uses the `<OBJECT>` or `<EMBED>` tag, and also allows you to address the movie by name using the QuickTime JavaScript commands.

```
<script language="javascript" type="text/javascript">

 QT_WriteOBJECT('MyMovie.mov' , '100%', '95%', '', 'SCALE', 'aspect',
'obj#ID', 'movieOBJ', 'emb#ID', 'movieEMBED') ;

</script>
```

The `AC_QuickTime.js` script provides three additional functions which use the same input syntax as `QT_WriteOBJECT`:

- `QT_GenerateOBJECTText`—generates the same output as `QT_WriteOBJECT`, but instead of inserting it into your HTML, it returns the output as a text string, allowing you to inspect it or modify it.
- `QT_WriteOBJECT_XHTML`—generates and inserts the `OBJECT` and `EMBED` tags in your file using strict XHTML syntax, and should be used if you are embedding QuickTime in XHTML rather than HTML.

- `QT_GenerateOBJECTText_XHTML`—generates the same output as `QT_WriteOBJECT_XHTML`, but instead of inserting it into your XHTML, it returns it as a text strting.

For more information on embedding QuickTime content in HTML, see *HTML Scripting Guide for QuickTime*.

Executing JavaScript Functions From QuickTime

Important: Starting with QuickTime 7.1.5, you can no longer issue javascript:// URLs or call JavaScript functions directly from within a QuickTime movie. This feature was removed from QuickTime for security reasons. If you need to call a JavaScript function from a movie, the recommended practice is to load an HTML page into an Iframe, and call the JavaScript function in the `ONLOAD` parameter of the Iframe page.

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: URLs directly to the browser as part of a QuickTime URL is no longer supported.

In order to execute a JavaScript function from a movie in QuickTime 7.1.5 or later, you must load an HTML page that calls the desired JavaScript function. One way to make the user experience as seamless as possible is to load an HTML page into a hidden Iframe, with a single JavaScript function defined in the `<head>` element, set to execute `onload`. This way, loading the page is essentially the same as executing the JavaScript function.

If you need to call a JavaScript function defined on page that loads the movie, call it from the Iframe page using `parent.FunctionName()` addressing syntax.

For example, suppose you have a JavaScript function called `MyFunction()` that you wish to call from QuickTime. In the past you could call it directly. Now, you must call it indirectly. Here's how:

1. In the page that contains `MyFunction()`, create an invisible Iframe, like this:

```
<iframe id="Iframe1" name="Iframe1"

frameborder="0" vspace="0" hspace="0"

marginwidth="0" marginheight="0" width="0" height="0"

src="blank.html">

</iframe>
```

2. Note that the Iframe needs an initial source URL. This example uses "`blank.html`", an html page that has no content:

```
<HTML>

<HEAD> <TITLE>blank.html</TITLE> </HEAD>

<BODY> </BODY>

</HTML>
```

3. Create a page intended to load into the Iframe. This page contains a single JavaScript function that calls `MyFunction()`. This new function, named `CallMyFunction()`, is called when the page loads, using the `onload` attribute of the `<BODY>` tag:

```
<HTML>

<HEAD>

<TITLE>callmyfunction.html</TITLE>

<Script language="javascript" type="text/javascript">

function CallMyFunction()

{  parent.MyFunction();  }

</Script>

</HEAD>

<BODY onload="CallMyFunction()">  </BODY>

</HTML>
```

4. Modify the URL called by the movie, from "`javascript:MyFunction();`" to "`<http://CallMyFunction.html> T<Iframe1>`"

   This changes the URL from `javascript:` to `http://` protocol, and loads the new page into the Iframe named "`Iframe1`". The new page calls `MyFunction()` in the parent page when it loads. For the user, the experience is the same as if the movie had called `MyFunction()` directly.

The following code example shows two HTML pages. The first page creates an Iframe, defines a function, `test()`, to be called from QuickTime, and embeds a movie. The second page, which will be loaded into the Iframe, defines a JavaScript function, `CallParentTest()`, that calls the `test()` function in the parent page. The `onload` attribute of the `BODY` tag is set to execute the `CallParentTest()` function when the second page loads.

Listing 1-2 Executing a JavaScript function from QuickTime using an invisible Iframe

```
<html>
```

```
    <head>

    <Title>calltest.html (Page That Executes JavaScript test() Function
Indirectly)</Title>

    <script language="javascript" type="text/javascript">

    function CallParentTest()

    {

     parent.test();

    }

    </script>



    <body onload="CallParentTest()">

    </body>

</html>



<html>

    <head>

    <Title><Page that contains in invisible iFrame>

    <script src="AC_QuickTime.js" language="javascript"
type="text/javascript"> </script>

    <script language="javascript" type="text/javascript">

    function test()

    {

     alert("JavaScript function called from QuickTime!");

    }

    </script>
```

```
    </head>



    <body>

    <h2>This Movie Calls JavaScript idirectly using an Iframe</h2>



    <iframe  id="Iframe1"  name="Iframe1" frameborder="0"  vspace="0"
hspace="0"  marginwidth="0"  marginheight="0" width="2"  height="2"
src="blank.html">

    </iframe>



    <script language="javascript" type="text/javascript">

    QT_WriteOBJECT('MyMovie.mov' , '320', '240', '', 'HREF',
'<Calltest.html> T<Iframe1>');

    </script>



    <p>Click on the display portion of the movie to execute the
JavaScript alert() function.

    </p>

    </body>

</html>
```

You can use this technique to execute any number of JavaScript functions from a movie. Just call each JavaScript function from its own HTML page with the page set to execute the function using the `onload` attribute of the `<body>` element. The JavaScript function can be defined entirely within the Iframe page, or it can call other JavaScript functions defined on the parent page using the `parent.functionName()` syntax.

Call the URL of the page with the target set to the name of the Iframe. Note that the syntax for specifying the URL includes the target Iframe: `'<url> T<frameID>'`. This is true whether the URL is invoked from an `HREF`, `QTNEXTn`, or `HOTSPOT` parameter, for example.

Important: A URL called from QuickTime cannot cross local/remote zone boundaries. If the movie is loaded using a remote protocol, such as `http://`, any URL called from the movie must also use a remote protocol (`http://` or `https:/` or `rtsp://`). A movie loaded via `http://` cannot call a `file:///` URL. Similarly, if the movie is loaded using a local `file:///` URL, it cannot call a remote URL (`http://`, for example).

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, npruntime, 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 for Macintosh also support a Cocoa interface. Internet Explorer for Windows allows JavaScript to interact with plug-ins using the COM interface. Current versions of Mozilla, Opera, and Safari for Windows and Macintosh support npruntime.

Support for LiveConnect was added to QuickTime in version 4.1. Support for COM, XPCOM, and Cocoa were added in QuickTime 6. Support for npruntime was added in QuickTime 7.1.6.

Supported Browsers and Operating Systems

The QuickTime plug-in is scriptable from all browsers that support the LiveConnect, XPCOM, Cocoa, npruntime, or COM interface. This includes all versions of Netscape and Mozilla for Windows and Macintosh, AOL 5 and later for Windows, Firefox, Opera, 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). Safari for Windows is also supported.

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 EnableJavaScript="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.

- If you are using QuickTime DOM events, set an `id` attribute for the `<EMBED>` tag as well, using a unique value.

The easiest way to perform these steps is to include the `AC_QuickTime.js` script and pass the `name` and `id` attributes, set to the same value, along with the movie url and other parameters.

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 name

```
<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>
```

You can dramatically shorten the previous code example by taking advantage of the `AC_QuickTime.js` file, as described in "Using JavaScript to Detect QuickTime." The shortened code is shown in Listing 1-4.

Listing 1-4 Enabling JavaScript Using AC_QuickTime.js

```
<script language="javascript" type="text/javascript">

    QT_WriteOBJECT('MyMovie.mov' , '320', '256', '',
'EnableJavaScript', 'True', 'emb#NAME' , 'movie1' , 'obj#id' ,
'movie1') ;

</script>
```

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 1-5 gives an example of JavaScript usage with QuickTime.

Listing 1-5 Using JavaScript to play, stop, and replace a QuickTime movie

```
<html>

 <head>

         <title>Simple QuickTime Movie Controls</title>



<script src="AC_QuickTime.js" language="JavaScript"
type="text/javascript">

</script>

 </head>



 <body >
```

```
        <P>
        This page uses JavaScript to control a QuickTime movie...
        </P>
<div align=center>
      <table>
           <tr>
           <td width=200>
           <script language="javascript" type="text/javascript">
           QT_WriteOBJECT('MyMovie.mov', '180','160', '',
           'obj#id', 'movie1', 'emb#name', 'movie1',
'enablejavascript', 'true');
           </script>
           <P> Movie1 </P>
           </td>

           <td width=200>
           <script language="javascript" type="text/javascript">
           QT_WriteOBJECT('MyOtherMovie.mov', '180','160', '',
           'obj#id', 'movie2', 'emb#name', 'movie2',
'enablejavascript', 'true');
           </script>
           <P> Movie2 </P>
           </td>
           </tr>
```

```
      </table>
</div>

 <P>Play and Stop Movies: <br>
 <a href="javascript:document.movie1.Play();">Play Movie1</a><br>
 <a href="javascript:document.movie1.Stop();">Stop Movie1</a><br>
 <a href="javascript:document.movie2.Play();">Play Movie2</a><br>
 <a href="javascript:document.movie2.Stop();">Stop Movie2</a><br>
 </P>

 <P>Replace One Movie with Another: <br>
 <a
href="javascript:document.movie1.SetURL('MyOtherMovie.mov');">movie1.Se
tURL(MyOtherMovie.mov)</a>
<br>
 <a
href="javascript:document.movie1.SetURL('MyMovie.mov');">movie1.SetURL(
MyMovie.mov)</a>
<br>
 <a
href="javascript:document.movie2.SetURL('MyOtherMovie.mov');">movie2.Se
tURL(MyOtherMovie.mov)</a>
<br>
<a
href="javascript:document.movie2.SetURL('MyMovie.mov');">movie2.SetURL(
MyMovie.mov)</a><br>
 </P>

 </body>
```

```
</html>
```

QuickTime DOM Events

The browser plug-ins for QuickTime 7.2.1 and later include the ability to emit Document Object Model (DOM) events. DOM events can be emitted when the plug-in is instantiated and ready to interact with JavaScript, at various points during a movie's loading process, when playback has begun, paused, or ended, or in the event of an error. JavaScript functions can be set to "listen" for particular DOM events. Whenever the DOM event occurs, the listener function is called.

This allows you to create web pages that detect events such as movie loading, playing, pausing, or ending, without having to create JavaScript timing loops or constantly poll the plug-in for status. It also allows you to execute JavaScript functions in response to various movie events without sending `javascript://` URLs from a movie.

The plug-in posts standard DOM events, so in browsers that support the W3C's DOM Level 2 Event Specification (http://www.w3.org/TR/DOM-Level-2-Events/) you use the `addEventListener()` method to monitor for the events.
Safari 3.0 and FireFox 2.0 are examples of browsers that support the W3C standards sufficiently to work with the methods described in this document.

Internet Explorer version 5 and later supports a slightly different method that is essentially equivalent: `attachEvent()`.

For a page to work correctly whether visitors are using browsers that support the W3C standards or using Internet Explorer, you must include JavaScript code for both the `addEventListener()` function and the `attachEvent()` function.

All the examples in this document use both methods and should work with all W3C-compliant browsers, and well as with Internet Explorer.

Important: The user must have QuickTime 7.2.1 or later installed in order to receive DOM events.

Summary of DOM Events

QuickTime can emit DOM events when the plug-in is instantiated and ready to interact with JavaScript, at various points during a movie's loading process, when playback has begun, paused, or ended, or in the event of an error. The following list shows the DOM events that can be listened for. Note that all QuickTime DOM events begin with the prefix "qt_" to prevent name space collisions.

- `qt_begin` — The plug in has been instantiated and can interact with JavaScript.
- `qt_loadedmetadata` — The movie header information has been loaded or created. The duration, dimensions, looping state, and so on are now known.
- `qt_loadedfirstframe` — The first frame of the movie has been loaded and can be displayed. (The frame is displayed automatically at this point.)
- `qt_canplay` — Enough media data has been loaded to begin playback (but not necessarily enough to play the entire file without pausing).
- `qt_canplaythrough` — Enough media data has been loaded to play through to the end of the file without having to pause to buffer, assuming data continues to come in at the current rate or faster. (If the movie is set to autoplay, it will begin playing now.)
- `qt_durationchange` — The media file's duration is available or has changed. (A streaming movie, a `SMIL` movie, or a movie with a `QTNEXT` attribute may load multiple media segments or additional movies, causing a duration change.)

- qt_load — All media data has been loaded.
- qt_ended — Playback has stopped because end of the file was reached. (If the movie is set to loop, this event will not occur.)
- qt_error — An error occurred while loading the file. No more data will be loaded.
- qt_pause — Playback has paused. (This happens when the user presses the pause button before the movie ends.)
- qt_play — Playback has begun.
- qt_progress — More media data has been loaded. This event is fired no more than three times per second.

  This event occurs repeatedly until the qt_load event or qt_error event. The last progress event may or may not coincide with the loading of the last media data. Use the progress function to monitor progress, but do not rely on it to determine whether the movie is completely loaded. Use the qt_load function in conjunction with the qt_progress function to monitor load progress and determine when loading is complete.
- qt_waiting — Playback has stopped because no more media data is available, but more data is expected. (This usually occurs if the user presses the play button prior to the qt_canplaythrough event. It can also occur if the data throughput slows during movie playback, and the buffer runs dry.)
- qt_stalled — No media has been received for approximately three seconds.
- qt_timechanged — The current time has been changed (current time is indicated by the position of the playhead).
- qt_volumechange — The audio volume or mute attribute has changed.

Enabling DOM Events

The QuickTime plug-in will emit DOM events only if this feature is explicitly enabled by setting postdomevents true in the <OBJECT> or <EMBED> tag.

In Internet Explorer you must also ensure that the <HEAD> element contains an <OBJECT> element for the “binary behavior” object (clsid CB927D12-4FF7-4a9e-A169-56E4B8A75598) that emits DOM events. The <OBJECT> tag that invokes the plug-in must include a style attribute that references the “binary behavior” object's ID, using the following syntax: style="behavior:url(#BinaryBehaviorID)".

For example, if the binary behavior object has the ID “qt_event_source”, the object tag that embeds the movie must include the attribute style='behavior:url(#qt_event_source)'. The necessary tags to perform these actions are automatically inserted by the AC_QuickTime.js script when the postdomevents parameter is set true.

Note: The ID for the <EMBED> and <OBJECT> tags must have unique values, but the <EMBED> NAME parameter can have the same value as the <OBJECT> ID to simplify movie addressing from JavaScript.

Listing 1-6 Enabling DOM Events Using AC_QuickTime.js

```
<script language="javascript" type="text/javascript">


    QT_WriteOBJECT('MyMovie.mov' , '320', '256', '',
'EnableJavaScript', 'True', 'postdomevents', 'True', 'emb#NAME' ,
'movie1' , 'obj#id' , 'movie1', 'emb#id', 'movie_embed1') ;
```

```
</script>
```

If you prefer not to use the `AC_QuickTime.js` script to create the `<OBJECT>` and `<EMBED>` tags, the following listing shows the syntax for creating the necessary tags manually.

Listing 1-7 Enabling DOM Events Manually

```
<object id="qt_event_source" classid="clsid:CB927D12-4FF7-4a9e-A169-
56E4B8A75598"
codebase="http://www.apple.com/qtactivex/qtplugin.cab#version=7,2,1,0"
></object>

<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"

codebase="http://www.apple.com/qtactivex/qtplugin.cab#version=7,2,1,0"

         width="320" height="256" type="video/quicktime" id="movie1"

         style="behavior:url(#qt_event_source);">



  <param name="src" value="MyMovie.mov" />         <param
name="postdomevents" value="true" />

       <embed

            src="MyMovie.mov"

            width="320" height="256"

            id="movie_embed1"

            name="movie1"

            postdomevents="true">

       </embed>

   </object>
```

Listening For DOM Events

In order to respond to DOM events, you need to create and register a “listener” function for each DOM event you are interested in.

For browsers that work with the W3C DOM Level 2 Event Specification, use the `addEventListener()` function; for Internet Explorer, use the `attachEvent()` function and add the prefix “`on`” to the event. The following listing gives an example.

Listing 1-8 Cross-platform event listener registration

```
<script language="javascript" type="text/javascript">

function myAddListener(obj, evt, handler, captures)

    {

        if ( document.addEventListener )

            obj.addEventListener(evt, handler, captures);

        else

            // IE

            obj.attachEvent('on' + evt, handler);

    }

</script>
```

Using DOM Events to Monitor Movie Loading

Here's an example of a web page that listens for the DOM event emitted periodically when a movie is loading, and displays the percent of the movie loaded so far. It also monitors for the movie loaded event, and overwrites the progress message with a "movie loaded" message when the movie is fully loaded.

Listing 1-9 Monitoring movie loading using DOM events

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/html140/DTD/loose.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

    <head>

    <script src="AC_QuickTime.js" language="JavaScript"
type="text/javascript">

    </script>



    <script language ="JavaScript" type="text/javascript">

 <!--

     /* define function that shows percentage of movie loaded */
```

```
        function showProgress()

        {

        var percentLoaded = 0 ;

        percentLoaded = parseInt((document.movie1.GetMaxTimeLoaded() /
document.movie1.GetDuration()) * 100);

        document.getElementById("loadStatus").innerHTML = 'Movie
loading: ' + percentLoaded + '% complete...';

        }

    /* define function that executes when movie loading is complete */

       function movieLoaded()

        {

        document.getElementById("loadStatus").innerHTML = "Movie
Loaded" ;

        }



    /* define function that adds another function as a DOM event
listener */

         function myAddListener(obj, evt, handler, captures)

        {

        if ( document.addEventListener )

            obj.addEventListener(evt, handler, captures);

        else

            // IE

            obj.attachEvent('on' + evt, handler);

       }
```

```
   /* define functions that register each listener */

    function RegisterListener(eventName, objID, embedID, listenerFcn)

    {

        var obj = document.getElementById(objID);

        if ( !obj )

            obj = document.getElementById(embedID);

        if ( obj )

            myAddListener(obj, eventName, listenerFcn, false);

    }



    /* define a single function that registers all listeners to call
onload */

    function RegisterListeners()

    {

       RegisterListener('qt_progress', 'movie1', 'qtmovie_embed',
showProgress);

       RegisterListener('qt_load', 'movie1', 'qtmovie_embed',
movieLoaded);

    }



 //-->

 </script>

    <title>JavaScript Movie Monitor</title>

 </head>
```

```
<body  onload="RegisterListeners()">

<div align=center>

<h2>Movie with JavaScript Progress/Load Monitors</h2>



<script language="javascript" type="text/javascript">

 QT_WriteOBJECT('MyMovie.mov' , '320', '256', '', 'EnableJavaScript',
'True',

'postdomevents', 'True', 'emb#NAME' , 'movie1' , 'obj#id' , 'movie1',
'emb#id',

 'qtmovie_embed') ;

</script>



        <p ID="loadStatus">

          MOVIE LOADING...

        </p>

   <p> <a href="javascript:document.movie1.Play();">Play</a> </p>

   <p> <a href="javascript:document.movie1.Stop();">Stop</a> </p>

</div>

</body>

</html>
```

Monitoring and Controlling Multiple Movies

You can control and monitor multiple movies on a single web page using JavaScript controls and DOM events. You need to give each movie a unique name in the `<EMBED>` tag, a matching `ID` in the `<object>` tag, as well as a unique `ID` in the `<EMBED>` tag.

Create and register DOM event listeners for each movie, using the `<EMBED>` and `<OBJECT>` IDs.

Create JavaScript controls for each movie, using the `<EMBED> NAME`, which is the same as the `<OBJECT> ID`.

For example, you might have two movies, the first with an `<OBJECT> ID` and `<EMBED> NAME` of `movie1`, and an `<EMBED> ID` of `movie_embed1`, and a second movie with an `<OBJECT> ID` and `<EMBED> NAME` of `movie2`, and an `<EMBED> ID` of `movie_embed2`. Register DOM event listeners for `movie1/movie_embed1` and `movie2/movie_embed2`, and create JavaScript controls for `movie1` and `movie2`.

The following listing is an example of a web page that monitors the load progress and load completion of two movies, and provides play and stop controls for each movie in JavaScript.

Listing 1-10 Web page that monitors and controls two movies using JavaScript and DOM events

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/html140/DTD/loose.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

    <head>

    <script src="AC_QuickTime.js" language="JavaScript"
type="text/javascript">

    </script>



    <script language ="JavaScript" type="text/javascript">

 <!--

    /* define function that shows percentage of movie 1 loaded */

        function showProgress()

        {

        var percentLoaded = 0 ;

        percentLoaded = parseInt((document.movie1.GetMaxTimeLoaded() /
document.movie1.GetDuration()) * 100);

        document.getElementById("loadStatus1").innerHTML = 'Movie
loading: ' + percentLoaded + '% complete...';

        }

   /* define function that executes when movie1 loading is complete */

        function movieLoaded()

        {
```

```
        document.getElementById("loadStatus1").innerHTML = "Movie
Loaded" ;

           }

    /* define function that shows percentage of movie 2 loaded */

        function showProgress2()

        {

        var percentLoaded = 0 ;

        percentLoaded = parseInt((document.movie2.GetMaxTimeLoaded() /
document.movie2.GetDuration()) * 100);

        document.getElementById("loadStatus2").innerHTML = 'Movie
loading: ' + percentLoaded + '% complete...';

        }

    /* define function that executes when movie2 loading is complete */

        function movieLoaded2()

        {

        document.getElementById("loadStatus2").innerHTML = "Movie
Loaded" ;

        }

   /* define function that adds another function as a listener for a
DOM event */

        function myAddListener(obj, evt, handler, captures)

        {

        if ( document.addEventListener )

            obj.addEventListener(evt, handler, captures);

        else

            // IE

            obj.attachEvent('on' + evt, handler);
```

```
        }


   /* define functions that register each listener */

    function RegisterListener(eventName, objID, embedID, listenerFcn)

    {

        var obj = document.getElementById(objID);

        if ( !obj )

            obj = document.getElementById(embedID);

        if ( obj )

            myAddListener(obj, eventName, listenerFcn, false);

    }



    /* define a single function that registers all listeners to call
onload */

    function RegisterListeners()

    {

     RegisterListener('qt_progress', 'movie1', 'qtmovie_embed',
showProgress);

     RegisterListener('qt_load', 'movie1', 'qtmovie_embed',
movieLoaded);

     RegisterListener('qt_progress', 'movie2', 'qtmovie_embed2',
showProgress2);

     RegisterListener('qt_load', 'movie2', 'qtmovie_embed2',
movieLoaded2);

    }

 //-->

   </script>
```

```
      <title>JavaScript Movie Controls and Monitors</title>

 </head>

<body  onload="RegisterListeners()">



<div align=center>

<h2>Two Movies with JavaScript Play/Stop Controls<br>and Progress/Load
Monitors</h2>

<script language="javascript" type="text/javascript">

QT_WriteOBJECT('MyMovie.mov', '360','256', '', 'obj#id', 'movie1',
'emb#id', 'qtmovie_embed', 'emb#name', 'movie1', 'postdomevents',
'true', 'enablejavascript', 'true');

</script>

  

<script language="javascript" type="text/javascript">

QT_WriteOBJECT('MyOtherMovie.mov', '360','256', '', 'obj#id', 'movie2',
'emb#id', 'qtmovie_embed2', 'emb#name', 'movie2', 'postdomevents',
'true', 'enablejavascript', 'true');

</script>



<table width="100%">

<tr>

<td align=center>

<P ID="loadStatus1">

MOVIE LOADING...

</p>

</td>

<td align=center>
```

```
<P ID="loadStatus2">
MOVIE LOADING...
</p>
</td>
</tr>

<tr>
<td align=center>
   <P> <a href="javascript:document.movie1.Play();">Play 1</a> </P>
</td>
<td align=center>
   <P> <a href="javascript:document.movie2.Play();">Play 2</a> </P>
</td>
</tr>
<tr>
<td align=center>
    <P> <a href="javascript:document.movie1.Stop();">Stop 1</a> </P>
</td>
<td align=center>
    <P> <a href="javascript:document.movie2.Stop();">Stop 2 </a> </P>
</td>
</tr>
</table>
```

```
</div>

</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).

GoToChapter

```
void GoToChapter(string chapterName)
```

Takes a chapter name and sets the movie's current time to the beginning of that chapter. See also: `GetChapterName` and `GetChapterCount` in movie properties.

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.

GetChapterCount

```
int GetChapterCount()
```

Returns the number of chapters in the movie.

GetChapterName

```
string GetChapterName(int)
```

Takes a chapter number and returns the chapter name.

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 strings

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.