Acrobat Multimedia JavaScript Reference

Version : Acrobat 6.0

ADOBE SYSTEMS INCORPORATED

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://partners.adobe.com

April 2004

Contents

Preface

Introduction

JavaScript is the cross-platform scripting language of Adobe Acrobat™. Through its JavaScript extensions, Acrobat exposes much of the functionality of the viewer and its plugins to the document author/form designer and developer. This document is the JavaScript reference to the Multimedia plugin. It documents the objects, properties and methods that can be used to control multimedia events.

Document Conventions

This document uses font conventions common to all Acrobat reference documents, and also uses a *quick bar* for many methods and properties to summarize their availability and usage restrictions.

Font Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions.

Font	Used for	Examples
monospaced	Paths and filenames	`C:\templates\mytmpl.fm`
	Code examples set off from plain text	These are variable declarations: `AVMenu commandMenu,helpMenu;`
monospaced bold	Code items within plain text	The **`GetExtensionID`** method ...
	Parameter names and literal values in reference documents	The enumeration terminates if **`proc`** returns **`false`**.
monospaced italic	Pseudocode	`ACCB1 void ACCB2 ExeProc(void) {` *`do something`* `}`
	Placeholders in code examples	`AFSimple_Calculate(`*`cFunction`*`,` *`cFields`*`)`

Font	Used for	Examples
blue	Live links to Web pages	The Acrobat Solutions Network URL is: http://partners/adobe.com/asn/
	Live links to sections within this document	See Using the SDK.
	Live links to other Acrobat SDK documents	See the *Acrobat Core API Overview*.
	Live links to code items within this document	Test whether an **`ASAtom`** exists.
bold	PostScript language and PDF operators, keywords, dictionary key names	The **setpagedevice** operator
	User interface names	The **File** menu
italic	Document titles that are not live links	*Acrobat Core API Overview*
	New terms	*User space* specifies coordinates for...
	PostScript variables	*filename* **deletefile**

Quick Bars

At the beginning of most property and method descriptions, a small table or *quick bar* provides a summary of the item's availability and usage recommendations.

This sample illustrates a quick bar, with descriptive column headings that are not normally shown.

Version or Deprecated	Save and Preferences	Security	Reader	Approval	Acrobat Pro
6.0	Ⓓ	Ⓢ	🅒	🅧	🅟

The following tables show the symbols that can appear in each column and their meanings

Column 1: Version or Deprecated

#.#

A number indicates the version of the software in which a property or method became available. If the number is specified, then the property or method is available only in versions of the Acrobat software greater than or equal to that number.

For Adobe Acrobat 6.0, there are some compatibility issues with older versions. Before accessing this property or method, the script should check that the forms version is greater than or equal to that number to ensure backward compatibility. For example:

```
if (typeof app.formsVersion !=
    "undefined" && app.formsVersion >= 6.0)
{
      // Perform version specific operations.
}
```

If the first column is blank, no compatibility checking is necessary.

Ⓧ

As the Acrobat JavaScript extensions have evolved, some properties and methods have been superseded by other more flexible or appropriate properties and methods. The use of these older methods are discouraged and marked by Ⓧ in the version column.

Column 2: Save and Preferences	
	Using this property or method dirties (modifies) the PDF document. If the document is subsequently saved, the effects of this method are saved as well.

Column 2: Save and Preferences	
Ⓟ	The preferences symbol indicates that even though this property does not change the document, it can permanently change a user's application preferences.

Column 3: Security	
	This property or method may only be available during certain events for security reasons (for example, batch processing, application start, or execution within the console).

Column 4: Availability in Adobe Reader	
	If the column is blank, the property or method is allowed in any version of the Adobe Reader.
	The property or method is not allowed in any version of the Adobe Reader.
🅐	The property or method is allowed only in version 5.1, or later, of the Adobe Reader, not in versions 5.05 or below.
🅕 🅒 🅢	The property or method can be accessed only in the Adobe 5.1 Reader depending on document rights. ● requires Advanced Forms Features rights ● requires the right to manipulate Comments. requires document Save rights.

Column 5: Availability in Adobe Acrobat Approval	
	If the column is blank, the property or method is allowed in Acrobat Approval.
	The property or method is not allowed in Acrobat Approval.

Column 6: Availability in Adobe Acrobat	
	If the column is blank, the property or method is allowed in Acrobat Std and Acrobat Pro.
	The property or method is available *only* in Acrobat Pro.

Other Sources of Information

Online Help

The Web offers a great many resources to help you with JavaScript in general as well as JavaScript for PDF. For example:

http://partners.adobe.com/asn/acrobat/—A listing of Acrobat resources for developers. This listing includes the following:

- http://www.adobe.com/support/forums/main.html—Adobe Systems, Inc. provides dedicated online support forums for all Adobe products, including Acrobat and Adobe Reader.
- http://www.adobe.com/support/database.html—In addition to the forums, Adobe maintains a searchable support database with answers to commonly asked questions.

http://forum.planetpdf.com/—This popular area of the PlanetPDF website contains discussion forums for beginners, developers, forms integrators, and more. Experts from around the world participate in the Planet PDF Forum.

PlanetPDF (http://www.planetpdf.com) has a section specifically devoted to JavaScript examples, called Planet PDF Developers.

References

Core JavaScript 1.5 Documentation

Complete documentation for JavaScript 1.5, the version used by Acrobat 6.0, is available on the web at http://devedge.netscape.com/central/javascript/.

Core JavaScript Guide, Netscape Communications Corporation. Part 1 of this document gives a conceptual overview of the core JavaScript language. This document is available in html format at http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/. Note: The rest of the document concerns Netscape's extensions to core JavaScript and are not applicable in the Acrobat environment.

Core JavaScript Reference, Netscape Communications Corporation. Parts 1 and 2 are references to the core JavaScript language. This document is available in html format at http://devedge.netscape.com/library/manuals/2000/javascript/1.5/reference/. Note: The rest of the document concerns Netscape's extensions to core JavaScript and are not applicable in the Acrobat environment.

Adobe Web Documentation

Acrobat JavaScript Scripting Reference, Technical Note # 5431, Version 6.0
http://partners.adobe.com/asn/acrobat/docs.jsp

PDF Reference, Fourth Edition, Version 1.5. The PDF Reference provides a description of the PDF file format and is intended primarily for application developers wishing to develop PDF producer applications that create PDF files directly.
http://partners.adobe.com/asn/tech/pdf/specifications.jsp

Acrobat Core API Overview, Technical Note #5190. Gives an overview of the objects and methods provided by the plug-in API of the Acrobat viewer. This document is available with the Adobe Acrobat Plug-ins SDK CD-ROM or on the Adobe Web site http://partners.adobe.com/asn/acrobat/docs.jsp.

Acrobat Core API Reference, Technical Note #5191. Describes in detail the objects and methods provided by the Acrobat viewer's plug-in API. This document is available with the Adobe Acrobat Plug-ins SDK CD-ROM or on the Adobe Web site http://partners.adobe.com/asn/acrobat/docs.jsp.

Acrobat Development Overview, Technical Note #5167. Describes how to develop Acrobat viewer plug-ins on the various platforms. http://partners.adobe.com/asn/acrobat/docs.jsp.

Programming Acrobat JavaScript using Visual Basic, Technical Note #5417. This document gives you the information you need to get started using the extended functionality of JavaScript from a Visual Basic programming environment. http://partners.adobe.com/asn/acrobat/docs.jsp

Acrobat Multimedia JavaScript Reference

Many of the JavaScript methods provided by Acrobat accept either a list of arguments as is customary in JavaScript, or alternatively, a single object argument with properties that contain the arguments. For example, these two calls are equivalent:

```
app.alert( "Acrobat Multimedia", 3);

app.alert({ cMsg: "Acrobat Multimedia", nIcon: 3});
```

The methods described in this multimedia JavaScript reference *do not* accept either argument format interchangeably, however. Use the exact argument format described for each method, whether it is a list of arguments or a single object argument containing various properties.

App Object

The Multimedia plug-in adds the following methods and properties to the App object.

App Object Properties

monitors

6.0				

The **`app.monitors`** property returns a Monitors Object, which is an array containing one or more Monitor objects representing each of the display monitors connected to the user s system. Each access to app.monitors returns a new, up to date copy of this array.

A Monitors object also has several methods that can be used to select a display monitor, or JavaScript code can look through the array explicitly. See the Monitors Object for details.

Type: Monitors Object *Access: R.*

Example

Count the number of display monitors connected to the user's system.

```
var monitors = app.monitors;
console.println("There are " + monitors.length
     + " monitor(s) connected to this system.");
```

App.media Object

The global **`app.media`** object defines properties and functions useful in multimedia JavaScript code.

Several of the properties of **`app.media`** are enumeration objects that list the values allowed in various properties. Note that future versions of Acrobat may add more such values, and JavaScript code should be prepared to encounter values other than the ones listed here. Similarly, JavaScript code may be run on an older version of Acrobat than it was designed for, in which case it will have to fall back to using the values available in that version.

App.media Object Properties

align

6.0				

The **`app.media.align`** property enumerates the values that may be found in the **`MediaSettings.floating.align`** property. The alignment is positioned relative to the window specified by the **`MediaSettings.floating.over`** property, see the values for **`app.media.over`**.

These values are

Value	Description
`app.media.align.topLeft`	position floating window at the top left corner
`app.media.align.topCenter`	position floating window at the top center
`app.media.align.topRight`	position floating window at the top right corner
`app.media.align.centerLeft`	position floating window at the center left
`app.media.align.center`	position floating window at the center
`app.media.align.centerRight`	position floating window at the center right
`app.media.align.bottomLeft`	position floating window at the bottom left corner
`app.media.align.bottomCenter`	position floating window at the bottom center
`app.media.align.bottomRight`	position floating window at the bottom right corner

Type: Object (enumeration) *Access: R.*

canResize

6.0				

`app.media.canResize` property enumerates the values that may be found in the **`MediaSettings.floating.canResize`** property, which specifies whether a floating window may be resized by the user.

These values are

Value	Description
`app.media.canResize.no`	may not be resized
`app.media.canResize.keepRatio`	may be resized only if aspect ration is preserved
`app.media.canResize.yes`	may be resized without preserving aspect ratio

Type: Object (enumeration) *Access: R.*

closeReason

6.0				

`app.media.closeReason` enumerates the values that may be found in the event.reason property for a Close event. These values are:

```
app.media.closeReason.general
app.media.closeReason.error
app.media.closeReason.done
app.media.closeReason.stop
app.media.closeReason.play
app.media.closeReason.uiGeneral
app.media.closeReason.uiScreen
app.media.closeReason.uiEdit
app.media.closeReason.docClose
app.media.closeReason.docSave
app.media.closeReason.docChange
```

See the afterClose and onClose events.

Type: Object (enumeration) *Access: R.*

defaultVisible

6.0				

The **`app.media.defaultVisible`** property is defined as **`true`**, which is the default value for **`MediaSettings.visible`**.

Type: Boolean *Access: R.*

ifOffScreen

6.0				

The **`app.media.ifOffScreen`** property enumerates the values allowed in a **`MediaSettings.floating.ifOffScreen`** property, which specifies what action should be taken if the floating window is positioned totally or partially offscreen.

These values and their descriptions are given in the table below:

Value	Description
`app.media.ifOffScreen.allow`	take no action
`app.media.ifOffScreen.forceOnScreen`	move and/or resize the window so that it is onscreen
`app.media.ifOffScreen.cancel`	cancel playing the media clip

Type: Object (enumeration) *Access: R.*

layout

6.0				

The **`app.media.layout`** property enumerates the values allowed in a **`MediaSettings.layout`** property.

The table below contains the values and their descriptions:

Value	Description
`app.media.layout.meet`	scale to fit all content, preserve aspect, no clipping, background fill
`app.media.layout.slice`	scale to fill window, preserve aspect, clip X or Y as needed
`app.media.layout.fill`	scale X and Y separately to fill window

Value	Description
`app.media.layout.scroll`	natural size with scrolling
`app.media.layout.hidden`	natural size with clipping
`app.media.layout.standard`	use player's default settings

Type: Object (enumeration) *Access: R.*

monitorType

6.0				

The **`app.media.monitorType`** property enumerates the values allowed in a **`MediaSettings.monitorType`** property.

The table below contains the values and their descriptions:

Value	Description
`app.media.monitorType.document`	The monitor containing the largest section of the document window
`app.media.monitorType.nonDocument`	The monitor containing the smallest section of the document window
`app.media.monitorType.primary`	Primary monitor
`app.media.monitorType.bestColor`	Monitor with the greatest color depth
`app.media.monitorType.largest`	Monitor with the greatest area (in pixels squared)
`app.media.monitorType.tallest`	Monitor with the greatest height (in pixels)
`app.media.monitorType.widest`	Monitor with the greatest width (in pixels)

Type: Object (enumeration) *Access: R.*

openCode

6.0				

The **`app.media.openCode`** enumerates the values that may be found in the code property of the return value from **`MediaPlayer.open()`**. The values are:

```
app.media.openCode.success
app.media.openCode.failGeneral
app.media.openCode.failSecurityWindow
app.media.openCode.failPlayerMixed
app.media.openCode.failPlayerSecurityPrompt
app.media.openCode.failPlayerNotFound
app.media.openCode.failPlayerMimeType
app.media.openCode.failPlayerSecurity
app.media.openCode.failPlayerData
```

Type: Object (enumeration) *Access: R.*

over

6.0				

The **app.media.over** property enumerates the values allowed in a **MediaSettings.floating.over** property, the value of which is used to align a floating window. See **app.media.align**.

Value	Description
`app.media.over.pageWindow`	align floating window relative to the document (page) window
`app.media.over.appWindow`	align floating window relative to the application window
`app.media.over.desktop`	align floating window relative to the full virtual desktop
`app.media.over.monitor`	align floating window relative to the (selected) monitor display screen

Type: Object (enumeration) *Access: R.*

pageEventNames

6.0				

The **app.media.pageEventNames** property enumerates the values that may be found in the **event.name** property for a page-level action. Event names that represent direct user actions are not included here. This enumeration is used to distinguish page-level actions from user actions. The values are:

```
app.media.pageEventNames.Open
app.media.pageEventNames.Close
app.media.pageEventNames.InView
app.media.pageEventNames.OutView
```

Type: Object (enumeration) *Access: R.*

Example

The a**`pp.media.pageEventNames`** can be used to distinguish between a page-level action and a direct user action. The script below is folder-level or document-level JavaScript that can be called from anywhere in a document.

```
function myMMfunction () {
     if ( app.media.pageEventNames[event.name] ) {
            console.println("Page Event: " + event.name);
            ...
     } else {
            console.println("User Generated Event: " + event.name);
            ...
    }
}
```

raiseCode

6.0				

The **`app.media.raiseCode`** property enumerates values that may be found in the error.raiseCode property when an exception is thrown. This property exists only when error.name is "RaiseError". Other values may be encountered in addition to these.

app.media.raiseCode.fileNotFound
app.media.raiseCode.fileOpenFailed

Type: Object (enumeration) *Access: R.*

Example

See the definition of **`app.media.getRenditionSettings()`** in the `media.js` file for examples of usage.

raiseSystem

6.0				

The **`app.media.raiseSystem`** property enumerates values that may be found in the error.raiseSystem property when an exception is thrown. This property exists only when error.name is "RaiseError".

```
app.media.raiseSystem.fileError
```

Other values may be added to the above property.

Type: Object (enumeration) *Access: R.*

Example

See the definition of **`app.media.getRenditionSettings()`** in the `media.js` file for examples of usage.

renditionType

6.0				

The **`app.media.renditionType`** property enumerates the values that may be found in **`Rendition.type`**. The values and their descriptions are given below.

Value	Description
`app.media.renditionType.unknown`	a type not known by this version of Acrobat
`app.media.renditionType.media`	a media rendition
`app.media.renditionType.selector`	a rendition selector

Type: Object (enumeration) *Access: R.*

status

6.0				

The **`app.media.status`** property enumerates the values that may be found in the **`event.media.code`** property for a Status event (see **`onStatus`**/**`afterStatus`**). Most of these values have additional information that is found in the event.text property. The values are:

Value	Description
`app.media.status.clear`	empty string - this status event, clears any message
`app.media.status.message`	general message
`app.media.status.contacting`	hostname being contacted
`app.media.status.buffering`	progress message or nothing
`app.media.status.init`	name of the engine being initialize
`app.media.status.seeking`	empty string

Along with the **`event.media.status`** code, there is also the **`event.media.text`**, a string that reflects the current status, as described above.

Type: Object (enumeration) *Access: R.*

See afterStatus and onStatus.

trace

6.0				

Set **`app.media.trace`** to **`true`** to print trace messages to the JavaScript console during player creation and event dispatching.

NOTE: **`app.media.trace`** is for test purposes only. Do not use this property in a PDF file that you publish. It will change in future versions of Acrobat.

Type: Boolean *Access: R/W.*

version

6.0				

`app.media.version` is the version number of the multimedia API defined in `media.js`, currently 6.0.

Type: Number *Access: R.*

windowType

6.0				

The **`app.media.windowType`** property enumerates the values allowed in a MediaSettings.windowType property. These values are given in the table below.

Value	Description
`app.media.windowType.docked`	docked to PDF page
`app.media.windowType.floating`	floating (popup) window
`app.media.windowType.fullScreen`	full screen mode

Type: Object (enumeration) *Access: R.*

App.media Object Methods

addStockEvents

6.0				

The **`app.media.addStockEvents()`** method adds stock event listeners to a MediaPlayer (see MediaPlayer Object) and sets **`player.stockEvents`** as a reference to these listeners for later removal.

If the optional **`annot`** is provided, then a reference to the annot is saved in **`MediaPlayer.annot`**. Later, when the player is opened with **`MediaPlayer.open()`**, stock event listeners will also be added to this annot, and **`annot.player`** will be set as a reference to the player.

Parameters

`player`	A required MediaPlayer Object
`annot`	(optional) A ScreenAnnot Object

Returns

Nothing

The stock event listeners provide standard Acrobat behavior such as focus handling.

If **`app.media.trace`** is **`true`**, then debug trace listeners are also included with the stock event listeners.

Use the **`removeStockEvents()`** method to remove event listeners that were added via **`addStockEvents()`**.

The **`app.media.createPlayer()`** and **`app.media.openPlayer()`** methods call **`addStockEvents()`** internally, so in most cases it is not necessary to call this method yourself.

alertFileNotFound

6.0				

The **`app.media.alertFileNotFound()`** method displays the standard file not found alert, with an optional don't show again checkbox.

Parameters

`oDoc`	**`oDoc`** is the document object the alert is associated with
`cFilename`	**`cFilename`** is the name of the missing file

`bCanSkipAlert`	(optional) If `bCanSkipAlert` is `true` and the user checks the checkbox, returns `true`, otherwise returns `false`. The default is `false`.

Returns

If `bCanSkipAlert` is `true`, returns `true` if checkbox is checked, otherwise returns `false`.

Example:

```
if ( !doNotNotify )
{
      var bRetn = app.media.alertFileNotFound(this, cFileClip, true );
      var doNotNotify = bRetn;
}
```

alertSelectFailed

6.0				

The `app.media.alertSelectFailed()` method displays the standard alert for a `rendition.select()` failure.

Parameters

`oDoc`	`oDoc` is the document object the alert is associated with
`oRejects`	(optional) If `oRejects` is provided, it should be an array of MediaReject Objects as returned by PlayerInfoList.select().
`bCanSkipAlert`	(optional) If `bCanSkipAlert` is `true` and the user checks the checkbox, returns `true`, otherwise returns `false`. The default is `false`.
`bFromUser`	(optional) `bFromUser` affects the wording of the alert message. It should be `true` if a direct user action triggered this code, or `false` if some other action such as selecting a bookmark triggered it. The default is `false`.

Returns

If `bCanSkipAlert` is `true`, returns `true` if checkbox is checked, otherwise returns `false`.

NOTE: When `rendition.select()` fails to find a usable player, and the `select()` parameter `bWantRejects` is set to `true`, the returned MediaSelection Object will contain an array of MediaReject Object, which can be passed to this method as the `oRejects` parameter. The `alertSelectFailed()` method will, in turn, ask the user to go to the web to download an appropriate player.

Example:

Displays "Cannot play media clip", with checkbox.

```
var bRetn = app.media.alertSelectFailed({
     oDoc: this,
     bCanSkipAlert: true
});
```

argsDWIM

6.0				

The **`app.media.argsDWIM`** method is a "Do What I Mean" function that is used by **`app.media.createPlayer()`**, **`app.media.openPlayer()`**, and **`app.media.startPlayer()`**. It fills in default values for properties that are not provided in the PlayerArgs Object, picking them out of the Event Object, so that these functions may be used as rendition action event handlers with no arguments or in custom JavaScript with explicit arguments. See **`app.media.createPlayer()`** for details of the PlayerArgs Object.

Parameters

`args`	The args is a PlayerArgs Object. See **`createPlayer()`** for details of the PlayerArgs object.

Returns

PlayerArgs Object

Example

See "Example 1" on page 30 for an example of usage.

canPlayOrAlert

6.0				

The **`app.media.canPlayOrAlert`** method determines whether any media playback is allowed and returns **`true`** if it is. If playback is not allowed, it alerts the user and returns **`false`**.

Parameters

`args`	The args is a PlayerArgs object. See `createPlayer()` for details of the PlayerArgs object.

Returns

Returns **`true`** if media playback is allowed, otherwise, this method returns **`false`**.

NOTE: The **`createPlayer()`** method calls this function before attempting to create a player. If you write your own code to substitute for **`createPlayer()`**, you can call **`canPlayOrAlert()`** to alert the user in situations where playback is not allowed, such as in multimedia authoring mode.

The only property in the args object that is used is doc, so you can use:

```
// There is a doc object in myDoc
if( app.media.canPlayOrAlert({ doc: myDoc })
/* OK to create player here */ ;
```

The above code displays "Cannot play media while in authoring mode", or other alerts, as appropriate.

computeFloatWinRect

6.0				

The **`app.media.computeFloatWinRect()`** method calculates and returns the rectangle in screen coordinates needed as specified by its parameters.

Parameters

`doc`	The document object for the document
`floating`	The floating parameters from the MediaSettings.floating object
`monitorType`	A number indicating which monitor to use. See the app.media.monitorType property.
`uiSize`	(optional) The user interface size given as an array of four numbers [w,x,y,z] representing the size, as returned by MediaPlayer.uiSize.

Returns

The rectangle in screen coordinates

Example:

```
var floating =
{
     over: app.media.over.monitor,
     align: app.media.align.center,
     canResize: app.media.canResize.no,
     hasClose: false,
     hasTitle: true,
     width: 400,
     height: 400
}
var rect = app.media.computeFloatWinRect
     (this, floating, app.media.monitorType.primary);
```

constrainRectToScreen

6.0				

The **`app.media.constrainRectToScreen()`** method returns a rectangle of screen coordinates, moved and resized if needed to place it entirely on some display monitor. If **`anchorPt`** is provided, and **`rect`** must be shrunk to fit, it shrinks proportionally toward **`anchorPt`** (which is an array of two numbers representing a point as [x,y]).

Parameters

`rect`	An array of four number representing screen coordinates of the desired rectangle
`anchorPt`	(optional) An array of two points [x,y] that is to be an anchor point

Returns

Returns a rectangle in screen coordinates.

createPlayer

6.0				

The **`app.media.createPlayer()`** creates a MediaPlayer Object without actually opening the player, using values provided in the **`args`** parameter. To open the player, call **`MediaPlayer.open()`**. You can combine these two steps into one by calling **`app.media.openPlayer()`** instead of **`createPlayer()`**.

If **`createPlayer()`** is called inside a rendition action (e.g. in custom JavaScript entered from the Actions tab in the Multimedia Properties panel), default values are taken from the action's Event Object. The **`args`** parameter is not required in this case unless you want to override the rendition action's values. The **`createPlayer()`** calls **`argsDWIM()`** to process the Event Object and **`args`** (see PlayerArgs Object) parameter.

Unless **`noStockEvents`** of the PlayerArgs Object is set to **`true`**, the MediaPlayer Object is equipped with stock event listeners which provide the standard behavior required to interact properly with Acrobat. Additional event listeners can be provided in the PlayerArgs object or may be added afterward with MeidaPlayer.events.add().

If **`args.annot.player`** is an open MediaPlayer, **`createPlayer()`** closes that player, which fires events.

Parameters

`args`	(optional) The **`args`** parameter is a PlayerArgs object. The parameter **`args`** is optional if createPlayer() is executed within a Rendition action with an associated rendition; in this case, the properties of **`args`** are populated by the defaults and by options selected in the UI. Otherwise, an **`args`** parameter is required, see documentation of the PlayerArgs object below for required properties of the object.

PlayerArgs Object

Property	Type	Description
`doc`	Object	The doc object of the document. Required if both annot and rendition are omitted, e.g. for URL playback.
`annot`	Object	A ScreenAnnot Object. Required for docked playback unless it is found in the Event Object or MediaSettings.page is provided. The new player is associated with the annot. If a player was already associated with the annot, it is stopped and closed.
`rendition`	Object	(optional) A Rendition Object (either a MediaRendition or a RenditionList). Required unless **`rendition`** found in the Event Object, or **`URL`** is present.
`URL`	String	Either **`URL`** or **`rendition`** is required, with **`URL`** taking precedence.
`mimeType`	String	(optional) Ignored unless **`URL`** is present. If **`URL`** is present, either **`mimeType`** or **`settings.players`**, as returned by **`app.media.getPlayers()`**, is required
`settings`	Object	(optional) A MediaSettings Object. Overrides the **`rendition`** settings.

Property	Type	Description
`events`	Object	(optional) An EventListener Object. Optional if stock events are used, added after stock events.
`noStockEvents`	Boolean	(optional) If **`true`**, do not use stock events. The default is **`false`**.
`fromUser`	Boolean	(optional) It should be **`true`** if a direct user action will trigger this code, or **`false`**, otherwise. The default depends on Event Object.
`showAltText`	Boolean	(optional) If **`true`**, show alternate text (see altText) if the media can't be played. The default is **`true`**.
`showEmptyAltText`	Boolean	(optional) If **`true`** and alternate text (see altText) is empty, show the alternate text as an empty box; if **`false`**, respond with an alert. The default value is **`true`** if **`fromUser`** is **`false`**, and **`false`** if **`fromUser`** is **`true`**.

Returns

MediaPlayer Object

Example 1

The following code is taken from `media.js`, it is the definition of **`openPlayer()`**, which uses **`createPlayer()`** in its definition.

```
app.media.openPlayer = function( args )
{
      var player = null;
      try
      {
             args = app.media.argsDWIM( args );

             player = app.media.createPlayer( args );
             if( player )
             {
                    var result = player.open();
                    if( result.code != app.media.openCode.success )
                    {
                           player = null;
                           app.media.alert
                                  ( "Open", args, { code: result.code } );
                    }
                    else if( player.visible )
                    player.setFocus();  // fires Focus event
             }
      }
```

```
        catch( e )
        {
              player = null;
              app.media.alert( 'Exception', args, { error: e } );
        }

        return player;
    }
```

Example 2

See the examples at the end of the description of **openPlayer()** for examples of PlayerArgs usage.

getAltTextData

6.0				

The **app.media.getAltTextData()** method returns a **MediaData object** (this is the same as the **MediaSettigs.data** object) which represents alternate text data for the given text. This **MediaData object** can be used to create a player to display the alternate text.

Parameters

cAltText	A string that is to be used as alternate text data

Returns

MediaData object

See **MediaSettings.data.**

Example

See the embedded example following **getAltTextSettings()**.

getAltTextSettings

6.0				

The **app.media.getAltTextSettings()** takes a PlayerArgs Object containing at least **settings**, **showAltText**, and **showEmptyAltText** properties, along with a selection object as returned by **rendition.select()**, and finds the first available alternate text rendition if there is one. It then creates and returns a new MediaSettings Object suitable for playback of the alternate text. Otherwise it returns **null**.

Parameters

`args`	A **`PlayerArgs Object`**, see PlayerArgs Object for more information on this object.
`selection`	A MediaSelection Object

Returns

MediaSettings Object or **`null`**

Example

This example plays back the alternate text of the rendition. The code plays back the alternate text in a screen annot, but can be modified for playback in a floating window.

```
var rendition = this.media.getRendition("myClip");
var settings = rendition.getPlaySettings();
var args = {
    settings: settings,
    showAltText: true,
    showEmptyAltText: true
};
var selection = rendition.select();
settings = app.media.getAltTextSettings( args, selection );

// You can also play custom alternate text by uncommenting the next line
// settings.data = app.media.getAltTextData("A. C. Robat");

// Uncomment the code below to obtain a floating window to playback
// the alternate text
/*
settings.windowType = app.media.windowType.floating
settings.floating = {
     canResize: app.media.canResize.keepRatio,
     hasClose: true,
     width: 400,
     height: 100
} */

// now define an args parameter for use with openPlayer, which will
// play the alternate text.
args = {
    rendition: rendition,
    annot: this.media.getAnnot({nPage: 0, cAnnotTitle:"myScreen"}),
    settings: settings
};
app.media.openPlayer(args);
```

getAnnotStockEvents

6.0				

The **`app.media.getAnnotStockEvents()`** method returns an Events Object containing the stock event listeners required in a screen annot for normal playback in Acrobat. The stock event listeners provide standard Acrobat behavior such as focus handling.

If **`app.media.trace`** is **`true`**, then debug trace listeners are also included with the stock event listeners.

Parameters

`settings`	A number corresponding to the windowType, see **`app.media.windowType`**.

Returns

Events Object

getAnnotTraceEvents

6.0				

The **`app.media.getAnnotTraceEvents()`** method returns an Events Object containing event listeners that provide a debugging trace as events are dispatched.

Parameters

None

Returns

Events Object

getPlayers

6.0				

The **`app.media.getPlayers()`** method returns a PlayerInfoList Object, which is an array of PlayerInfo Objects representing the available media players.

The PlayerInfoList may be filtered using its **`select()`** method, and it may be used in the **`settings.players`** property when creating a media player with **`createPlayer()`**.

See PlayerInfoList Object and PlayerInfo Object for more details.

Parameters

cMimeType	(optional) An optional MIME type such as "**audio/wav**". If **cMimeType** is omitted, the list includes all available players. If **cMimeType** is specified, the list includes only players that can handle that MIME type.

Returns

PlayerInfoList Object

Example 1

List MP3 players to the debug console.

```
var mp = app.media.getPlayers("audio/mp3")
for ( var i = 0; i < mp.length; i++) {
    console.println("\nmp[" + i + "] Properties");
    for ( var p in mp[i] ) console.println(p + ": " + mp[i][p]);
}
```

Example 2

Choose any player that can play Flash media by matching the MIME type. The code assumes the code below is executed as a Rendition action with associated rendition (so no arguments for **createPlayer()** are required).

```
var player = app.media.createPlayer();
player.settings.players
     = app.media.getPlayers( "application/x-shockwave-flash" );
player.open();
```

getPlayerStockEvents

6.0				

The **app.media.getPlayerStockEvents()** returns a Events Object containing the stock event listeners required in a media player for normal playback in Acrobat. The stock event listeners provide standard Acrobat behavior such as focus handling.

Use **MediaPlayer.events.add()** to add these stock events to a media player.

The **app.media.createPlayer()** and **app.media.openPlayer()** methods automatically call **getPlayerStockEvents()** internally, so it is not necessary to call this method yourself unless you're writing code that sets up all event listeners explicitly.

If **app.media.trace** is **true**, then debug trace listeners are also included with the stock event listeners.

Parameters

`settings`	A MediaSettings Object

Returns

Events Object

getPlayerTraceEvents

6.0				

The **`app.media.getPlayerTraceEvents()`** method returns an Events Object containing event listeners that provide a debugging trace as events are dispatched.

Parameters

None

Returns

Events Object

getRenditionSettings

6.0				

The **`app.media.getRenditionSettings()`** method calls **`Rendition.select()`** to get a MediaSelection Object, then MediaSelection.rendition.getPlaySettings() to get a MediaSettings Object for playback. If either of these fails, it calls the **`getAltTextSettings()`** method to get a MediaSettings Object for alternate text playback. Finally, it returns the resulting MediaSettings Object, or **`null`** if **`getAltTextSettings()`** returned **`null`** (i.e. alt text was not specified or not allowed).

Parameters

`args`	A PlayerArgs Object, see PlayerArgs Object for more information on this object.

Returns

MediaSettings Object or **`null`**

Example

See Example 3 following the **`openPlayer()`** method.

getURLData

6.0				

The **app.media.getURLData()** returns a **MediaData object** which represents data to be retrieved for a URL and optional MIME type. This **MediaData object** can be used to create a player which will access data from that URL. See **MediaSettings.data** for more information on the **MediaData object**.

Parameters

cURL	The URL form which media data is to be retrieved.
cMimeType	(optional) The MIME type of the data.

Returns

MediaData object

Example

The following example retrieves a media clip from the Internet and plays it in a floating window.

```
var myURLClip = "http://www.mywebsite.com/myClip.mpg";
var args = {
     URL: myURLClip,
     mimeType: "video/x-mpg",
     doc: this,
     settings: {
            players: app.media.getPlayers("video/x-mpg"),
            windowType: app.media.windowType.floating,
            data: app.media.getURLData(myURLClip,"video/x-mpg"),
            floating: { height: 400, width: 600 },
     }
}
app.media.openPlayer(args);
```

getURLSettings

6.0				

The **app.media.getURLSettings()** method takes a PlayerArgs Object which contains a **settings** property and returns a MediaSettings Object suitable for playback of a URL. The settings property must contain a **URL** property and may contain a **mimeType** property. It may also contain additional settings which are copied into the resulting settings object.

Parameters

`args`	A PlayerArgs Object, see PlayerArgs Object for more information on this object.

Returns

MediaSettings Object

Example 1

Same example as above. Basically, **`getURLSettings()`** calls **`getURLData()`** and inserts the return MediaData object into the **`data`** property into the **`setting`**, which it then returns.

```
var myURLClip = "http://www.mywebsite.com/myClip.mpg";
args = {
     URL: myURLClip,
     mimeType: "video/x-mpg",
     doc: this,
     settings:
     {
            players: app.media.getPlayers("video/x-mpg"),
            windowType: app.media.windowType.floating,
            floating: { height: 400, width: 600 }
     }
};
settings = app.media.getURLSettings(args)
args.settings = settings;
app.media.openPlayer(args);
```

Example 2

The example below is a custom Keystroke action of a combo box. The combox is a simple playlist of streamed audio/video web sites. The export value of each element in the list has the form "URL,mimeType", for example

```
http://www.mySite.com/streaming/radio.asx,video/x-ms-asx
```

The script below first splits the export value into an array of length 2, the first element is the URL, the second is the mimeType. Any video will be shown in the screen annot "myScreen"; otherwise, only audio is heard.

```
if (!event.willCommit)
{
    var aURLMime = event.changeEx.split(",")
    console.println("aURLMime[0] = " + aURLMime[0]);
    console.println("aURLMime[1] = " + aURLMime[1]);
    var args = {
        annot:this.media.getAnnot({ nPage:0,cAnnotTitle: "myScreen" }),
        URL: aURLMime[0],
        mimeType: aURLMime[1],
        doc: this,
        settings: {
           players: app.media.getPlayers(aURLMime[1]),
           windowType: app.media.windowType.docked,
        }
    };
    settings = app.media.getURLSettings(args);
    args.settings = settings;
    var player = app.media.openPlayer(args);
}
```

getWindowBorderSize

6.0				

The **`app.media.getWindowBorderSize()`** method returns an array of four numbers representing the size in pixels of the left, top, right, and bottom borders that would be used for a floating window with the properties specified in the parameters.

The **`hasTitle`** and **`hasClose`** parameters are booleans, and **`canResize`** may be any of the values in **`app.media.canResize`**.

These parameters have the same names as properties of a **`MediaSettings.floating`** object, so you can simply pass in a floating object as a single parameter:

```
var size = doc.media.getWindowBorderSize( settings.floating );
```

Parameters

`hasTitle`	(optional) The default is **`true`**
`hasClose`	(optional) The default is **`true`**
`canResize`	(optional) The default is **`app.media.canResize.no`**

Returns

An array of numbers of length 4

openPlayer

6.0				

The **`app.media.openPlayer()`** calls **`app.media.createPlayer()`** to create a MediaPlayer Object, and then calls **`MediaPlayer.open()`** to open the actual player.

This function fires several events which may include **`Ready`** (see **`onReady`** and **`afterReady`**), **`Play`** (see **`onPlay`** and **`afterPlay`**) and **`Focus`** (see **`onFocus`** and **`afterFocus`**). See also the EventListener Object object for a general description of these events.

The method alerts the user and returns **`null`** on failure. Does not throw exceptions.

Parameters

`args`	(optional) The args is a PlayerArgs object. See PlayerArgs Object.

Returns

A MediaPlayer Object or **`null`** on failure

Example 1

The following is a minimal example. This is a Custom JavaScript from the Actions tab in the Multimedia Properties panel of a Screen Annot. To override the parameters specified by the UI of the Screen Annot, the **`args`** parameter is passed.

```
app.media.openPlayer();
```

Override **`settings.repeat`**: if repeat is set to 1, change it to 2; if not 1, set to 1.

```
var nRepeat =
     ( event.action.rendition.getPlaySettings().repeat == 1 ) ? 2 : 1;
var args = { settings: { repeat: nRepeat } };
app.media.openPlayer(args);
```

See the Event Object for an explanation of **`event.action.rendition`**. The above example also uses **`Rendition.getPlaySettings()`** to access the settings associated with the rendition to be played (the one associated with the Screen Annot).

Example 2

The following script is executed from a mouse up action of a form button. It plays a docked media clip in a ScreenAnnot.

```
app.media.openPlayer({
     rendition: this.media.getRendition( "myClip" ),
     annot: this.media.getAnnot( {nPage:0,cAnnotTitle:"myScreen"} ),
     settings: { windowType: app.media.windowType.docked }
});
```

Example 3

This is a Custom JavaScript from the Actions tab in the Multimedia Properties of a Screen Annot. The user click on the annot, and a randomly chosen movie clip is played.

```
// these are placed at the top level of the document JavaScripts
var myRenditions = new Array();
myRenditions[0] = "myClip1";
myRenditions[1] = "myClip2";
myRenditions[2] = "myClip3";

// this code is a Custom JavaScript of a ScreenAnnot. All renditions
// are docked and are played in the ScreenAnnot.
var l = myRenditions.length;
randomIndex = Math.floor( Math.random() * l ) % l;
```

```
var rendition = this.media.getRendition(myRenditions[randomIndex]);
var settings = app.media.getRenditionSettings({ rendition: rendition });

var args = { rendition: rendition, settings: settings }
app.media.openPlayer(args);
```

removeStockEvents

6.0				

The **`app.media.removeStockEvents()`** method removes any stock event listeners from a MediaPlayer Object and from any associated ScreenAnnot Object, and deletes the **`player.stockEvents`**, **`player.annot`**, **`annot.stockEvents`**, and **`annot.player`** properties. This undoes the effect of a previous **`addStockEvents()`** call.

Parameters

`player`	A MediaPlayer Object

Returns

Nothing

startPlayer

6.0				

The **`app.media.startPlayer()`** method checks whether an annot is provided in the PlayerArgs Object and the annot already has a player open. If so, it calls player.play() on that player to start or resume playback. If not, it calls **`app.media.openPlayer()`** to create and open a new MediaPlayer Object. See **`openPlayer`** for more details.

NOTE: **`app.media.startPlayer()`** is the default Mouse Up action when you use the Acrobat user interface to create a multimedia annot and rendition and don't specify any custom JavaScript.

Parameters

`args`	(optional) The args is a PlayerArgs object. See PlayerArgs Object.

Returns

A MediaPlayer Object or **`null`** on failure

Example

Start a screen annot from a form button.

```
var args = {
     rendition: this.media.getRendition( "myClip" ),
     annot: this.media.getAnnot({ nPage: 0, cAnnotTitle: "myScreen" }),
};
app.media.startPlayer(args);
```

Doc Object

The Multimedia plug-in adds the following properties and methods to the Doc object.

Doc Object Properties

media

6.0				

Each document has its own **`doc.media`** object, which contains properties that are specific to a particular document. **`doc.media`** also contains methods that apply to a document. The section on the Doc.media Object contains the documentation of the properties and methods of this object.

Type: DocMedia Object *Access: R/W.*

innerAppWindowRect

6.0				

This property returns the rectangle, an array of screen coordinates, for the Acrobat inner application window The application window is available as an outer rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

Type: Array of Numbers *Access: R.*

Example:

```
this.innerAppWindowRect
```

See also innerDocWindowRect, outerAppWindowRect and outerDocWindowRect.

innerDocWindowRect

6.0				

This property returns the rectangle, an array of screen coordinates, for the Acrobat inner document window. The document window is also available as an outer rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

These rectangles may differ quite a bit on different platforms. For example, on Windows, the doc window is always inside the app window, while on the Macintosh they are the same.

Type: Array of Numbers *Access: R.*

See also innerAppWindowRect, outerAppWindowRect, outerDocWindowRect and pageWindowRect.

outerAppWindowRect

6.0				

This property returns the rectangle, an array of screen coordinates, for the Acrobat outer application window The application window is available as an inner rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

Type: Array of Numbers *Access: R.*

See also innerAppWindowRect, outerDocWindowRect, outerDocWindowRect and pageWindowRect.

outerDocWindowRect

6.0				

This property returns the rectangle, an array of screen coordinates, for the Acrobat outer document window. The document window is available as an inner rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

These rectangles may differ quite a bit on different platforms. For example, on Windows, the doc window is always inside the app window, while on the Macintosh they are the same.

Type: Array of Numbers *Access: R.*

See also innerAppWindowRect, outerDocWindowRect, outerAppWindowRect and pageWindowRect.

pageWindowRect

6.0				

This property returns the rectangle, an array of screen coordinates, for the Acrobat page view window. The page view window is the area inside the inner document window in which the actual PDF content is displayed.

Type: Array of Numbers *Access: R.*

See also innerAppWindowRect, outerDocWindowRect, outerAppWindowRect and outerDocWindowRect.

Doc.media Object

The **doc.media** of each document contains multimedia properties that are specific to that document, and methods that apply to the document.

Doc.media Object Properties

canPlay

6.0				

The **doc.media.canPlay** property indicates whether multimedia playback is allowed for a document. Playback depends on the user's Trust Manager preferences and other factors. For example, playback is not allowed in authoring mode.

doc.media.canPlay returns an object that contains both a yes/no indication and a reason why playback is not allowed, if that is the case.

Type: Object *Access: R.*

If playback is allowed, then **canPlay.yes** exists to indicate this. (It is an empty object, but it may contain other information in the future.) You can make a simple test like this:

```
if( doc.media.canPlay.yes )
{
     // We can play back multimedia for this document
}
```

If playback is not allowed, **canPlay.no** object exists instead. As with **canPlay.yes**, you can simply test for the existence of **canPlay.no**, or you can look inside it for information about why playback is not allowed. At least one of these properties or other properties that may be added in the future will exist within **`canPlay.no`**:

Properties of canPlay.no

Property	Description
`authoring`	can't play when in authoring mode
`closing`	can't play because the document is closing
`saving`	can't play because the document is saving
`security`	can't play because of security settings
`other`	can't play for some other reason

In addition, **`canPlay.canShowUI`** indicates whether any alert boxes or other user interface are allowed in response to this particular playback rejection.

Example:

```
var canPlay = doc.media.canPlay;
if( canPlay.no )
{
     // We can't play, why not?
     if( canPlay.no.security )
     {
          // The user's security settings prohibit playback,
          // are we allowed to put up alerts right now?
          if( canPlay.canShowUI )
               app.alert( "Security prohibits playback" );
          else
               console.println( "Security prohibits playback" );
     }
     else
     {
          // Can't play for some other reason, handle it here
     }
}
```

Doc.media Object Methods

deleteRendition

6.0				

The **`doc.media.deleteRendition()`** method deletes the named Rendition from the document. The Rendition is no longer accessible with JavaScript. It does nothing if the Rendition is not present.

Parameters

`cName`	**`cName`**, a string, is the name of the Rendition.

Returns

Nothing

Example

```
this.media.deleteRendition("myMedia");
if ( this.media.getRendition("myMedia") == null)
     console.println( "Rendition successfully deleted" );
```

getAnnot

6.0				

`Doc.media.getAnnot()` looks for and returns a ScreenAnnot Object in the document by page number and either name or title, or returns **`null`** if there is no matching ScreenAnnot. If both name and title are specified, both must match.

Parameters

`args`	An object containing the properties to be passed to this method. The properties are described below.

Properites of args

`nPage`	The page number (base 0) on which the Annot resides
`cAnnotName`	(optional) The name of the ScreenAnnot. **NOTE:** **`cAnnotName`** is never used in pdf generated by Acrobat.
`cAnnotTitle`	(optional) The title of the ScreenAnnot

NOTE: The parameters for this method must be passed as an object literal, and not as an ordered listing of parameters.

Returns

ScreenAnnot Object

Example

The Acrobat user interface allows you to specify the title for a ScreenAnnot but not its name, so a typical use of **`getAnnot`** would be:

```
var annot= myDoc.media.getAnnot
     ({ nPage: 0,cAnnotTitle: "My Annot Title" });
```

See the example following **`getRendition()`** for an additional example.

getAnnots

6.0				

The **`doc.media.getAnnots()`** method returns an Array of all the ScreenAnnot Objects on the specified page of the document, or all the ScreenAnnot Objects on all pages of the document if **`nPage`** is omitted. The array is empty if there are no such ScreenAnnots.

Parameters

`nPage`	The page number (base 0) on which the Annots reside

Returns

Array of ScreenAnnot Objects

Example

Get a listing of the ScreenAnnots on page 0, then play a media clip in a ScreenAnnot randomly chosen from the list.

```
var annots = this.media.getAnnots({ nPage: 0 });
var rendition = this.media.getRendition("myClip");
var settings = { windowType: app.media.windowType.docked }
var l = annots.length
var i = Math.floor( Math.random() * l ) % l
var args = { rendition:rendition, annot:annots[i], settings:settings };
app.media.openPlayer( args );
```

getRendition

6.0				

`doc.media.getRendition()` looks up a Rendition in the document by name and returns it, or returns **`null`** if there is no Rendition with that name.

Parameters

`cName`	**`cName`**, a string, is the name of the Rendition.

Returns

Rendition Object

Example

The following script is executed from a mouse up action of a form button. It plays a docked media clip in a ScreenAnnot.

```
app.media.openPlayer({
     rendition: this.media.getRendition( "myClip" ),
     annot: this.media.getAnnot( {nPage:0,cAnnotTitle:"myScreen"} ),
     settings: { windowType: app.media.windowType.docked }
});
```

newPlayer

6.0				

The **`doc.media.newPlayer()`** method creates and returns a MediaPlayer Object. The **`args`** parameter must contain a **`settings`** property and optionally can contain an **`events`** property. It can also contain any number of additional user-defined properties. All the properties of **`args`** are copied into the new MediaPlayer Object. This is a shallow copy: The properties of **`args`** are copied into the new player, but if any of those properties are objects themselves, those objects are shared between **`args`** and the new player.

The **`newPlayer()`** method creates a bare-bones player which does not have any of the standard event listeners required for standard Acrobat media player behavior. Use **`app.media.addStockEvents()`** to add the necessary event listeners.

In most cases it is better to use **`app.media.createPlayer()`** to create a media player instead of **`doc.media.newPlayer()`**. The **`createPlayer()`** sets up the standard event listeners and other player properties automatically. If you do call **`newPlayer()`** directly, the source code for **`createPlayer()`** in `media.js` should be reviewed for sample code.

Parameters

`args`	**`args`** is a PlayerArgs object. See PlayerArgs Object.

Returns

MediaPlayer Object

Example:

See **`Events.dispatch()`** for a rough example.

Event Object

Whenever an event fires and is dispatched to an event listener, an **`Event object`** is passed as a parameter to the event listener. This object is similar to the **`Event object`** used elsewhere in Acrobat, and it has the properties listed below.

For all events, these properties are the same as in other Acrobat Event objects:

```
modifier
name
shift
target
targetName
type
```

Multimedia Event objects fired by rendition actions (e.g. in custom JavaScript entered from the Actions tab in the Multimedia Properties panel) also include these properties:

`action.annot`	The Screen Annotation for this event (See ScreenAnnot Object)
`action.rendition`	The Rendition for this event (See Rendition Object)

Multimedia Event objects that have been dispatched by the standard multimedia event dispatcher also include these properties. These are not present if you provide your own **`events.dispatch()`** method:

`media.doc`	The document, same as `target.doc`
`media.events`	The events object, same as `target.events`
`media.id`	A copy of `event.name` with spaces removed

Individual events may have additional properties; see the description of each EventListener Object method for details.

An event method called by the standard event dispatcher may set either of these properties to stop further event dispatching:

```
stopDispatch
stopAllDispatch
```

To stop the current event from being dispatched to any remaining event listeners, an event method can set **`event.stopDispatch`** to **`true`**. If this is done in an "on" event method, no more "on" methods will be called for the event, but "after" methods will still be called. If you set **`event.stopAllDispatch`**, then no more event methods of either type will be called. Read about the EventListener Object for a description of the "on" and "after" event listeners

Events Object

A multimedia Events object (whose constructor is app.media.Events) is a collection of event listener objects. The events property of a MediaPlayer Object or a ScreenAnnot Object is an Events object.

Example:

This following is executed as rendition action

```
console.println("Ready to play \"" + event.action.rendition.uiName
    +"\" from screen annot \"" + event.targetName + "\".");
// Create a simple app.media.Events object
var events = new app.media.Events({
     // The Event object is passed as a parameter to all event
     // listeners, this is a the parameter "e" below/
     // Called immediately during a Play event:
     onPlay: function( e ) { console.println( "onPlay: media.id = "
            + e.media.id  ); },
     // Called during idle time after the Play event:
     afterPlay: function() { console.println( "afterPlay" ); },
});
var player = app.media.openPlayer({ events: events });
```

Events Object Methods

add

6.0				

Adds any number of EventListener Objects to the dispatch table for this Events Object. Any previous listeners are preserved, and when an event is fired, all matching listener methods are called.

The standard event dispatcher first calls any **`onEveryEvent`** methods in the order they were added, then calls any "on" events (see the description of "on" and "after" events in the introductory paragraphs to EventListener Object) for the specific event being dispatched, also in the order they were added. Finally, it sets a very short timer (one millisecond) to call any "after" events. When that timer fires, the "after" events are called in the same order described for on events.

NOTE: If you try to add the same event listener twice, the second attempt is ignored.

If you add an event listener from inside an event method, the new listener's methods will be called as part of the dispatching for the current event.

Parameters

Any number of parameters, each one an EventListener Object

Returns

Nothing

Example:

```
// Add an event listener for the onPlay event, here, player is a
// MediaPlayer object.
player.events.add
({
    onPlay: function() { console.println( "onPlay" ); }
});
```

See also Events.**`remove`**.

dispatch

6.0				

When a MediaPlayer fires an event, the Multimedia plug-in creates an Event Object and calls **`MediaPlayer.events.dispatch(event)`**. Similarly, a ScreenAnnot calls **`ScreenAnnot.events.dispatch(event)`**.

The dispatch method is the only part of the event dispatching system that the Acrobat Multimedia plugin calls directly. You can substitute your own, entirely different event dispatching system by providing your own **`MediaPlayer.events`** object with its own **`dispatch()`** method.

The **`dispatch()`** method is responsible for calling each of the event listeners associated with the event, as identified by **`oMediaEvent.name`**. In most cases, a PDF file will not provide its own **`dispatch()`** method but will use the standard event dispatching system.

Parameters

`oMediaEvent`	A Event Object

Returns

Nothing

If you write your own dispatch() method, note that **`oMediaEvent.name`** may contain spaces. The standard **`dispatch()`** method makes a copy of **`oMediaEvent.name`** in **`oMediaEvent.media.id`** with the spaces removed, to allow the name to be used directly as part of a JavaScript event method name.

Also, note that if you write your own **`dispatch()`**, it will be called synchronously when each event occurs, and any processing you do will be subject to the same limitations as described for "on" event methods in the EventListener section (see EventListener Object). In particular, it cannot make any calls to a MediaPlayer Object nor do anything that can indirectly cause a MediaPlayer method to be called. See the source code for the standard **`dispatch()`** in `media.js` for a way to work around this using a timer.

The **`dispatch()`** method is not usually called directly from JavaScript code, although it can be.

Example:

```
// Create a new media player with a custom event dispatcher.
// This is an advanced technique that would rarely be used in
// typical PDF JavaScript.
var player = doc.media.newPlayer(
{
     events:
     {
            dispatch: function( e )
            {
                   console.println( 'events.dispatch' + e.toSource() );
            }
     }
});
// Synthesize and dispatch a Script event, as if one had been
// encountered while the media was playing. With the standard event
// dispatcher, this will call any and all event listeners that have been
// added for this event. With the custom dispatcher above, it will log a
// message to the console.
var event = new Event;
event.name = "Script";
event.media = { command: "test", param: "value" };
player.events.dispatch( event );
```

remove

6.0				

The method removes one or more event listeners that were previously added with **`Events.add()`**. If you use an object literal directly in **`Events.add()`**, you will not be able to remove that listener using **`Media.remove()`** because there is no way to pass a reference to the same object. If you want to be able to remove an event listener, pass it to **`add()`** in a variable instead of an object literal, so that you can pass the same variable to **`remove()`**, as in the example below.

The remove() method may be called from inside an event method to remove any event listener, even the listener that the current event method is part of. The current event method continues executing, but no other event methods in the same event listener object will be called.

Parameters

Any number of parameters, each one an EventListener Object

Returns

Nothing

Example:

Assume **`player`** is a MediaPlayer object.

```
var listener = { afterStop: function() { app.alert("Stopped!"); } }
player.events.add( listener );        // add listener
.....
player.events.remove( listener );     // later, remove it
```

EventListener Object

An EventListener object is a collection of event method functions along with optional local data. Event method names begin with “on” or “after” followed by the event name, e.g. **`onPause`** or **`afterPause`**. When an event is dispatched, matching “on” event methods are called immediately, and matching “after” event methods are called a short while later, at the next idle time.

There are severe restrictions on what an “on” event method can do. In particular, an “on” event method for a MediaPlayer cannot call any of that MediaPlayer’s methods, nor can it call any other Acrobat method that may indirectly cause a method of the MediaPlayer to be called. For example, an “on” method must not close the document, save it, change the active page, change the focus, or anything else that may eventually call a method of the MediaPlayer.

An “after” event method does not have these restrictions. For most purposes, “after” event method are more versatile. Use an “on” event method only when the event must be processed synchronously at the time that it occurs, such as an **`onGetRect()`** method.

A note about reentrancy: “on” event methods are never reentered, but “after” event methods may be reentered.

Inside an event method, **`this`** is the event listener object. The document is available in **`event.media.doc`**, and the event target (MediaPlayer or ScreenAnnot) is in **`event.target`**.

`Events.add()` installs EventListener objects for dispatching, **`Events.dispatch()`** dispatches an event to the matching event methods, and **`Events.remove()`** removes EventListener objects from the dispatch table.

Example:

```
// Create a simple MediaEvents object
var events = new app.media.Events
({
     // Called immediately during a Play event:
     onPlay: function() { console.println( "onPlay" ); },

     // Called during idle time after the Play event:
     afterPlay: function() { console.println( "afterPlay" ); },
});
var player = app.media.createPlayer({events: events});
player.events.add({
     afterPlay: function( e ) {
            app.alert("Playback started, doc.URL = " + e.media.doc.URL );
     }
});
player.open();
```

EventListener Object Methods

The events listed here are specific to multimedia. In addition to these events, a ScreenAnnot may receive the standard events used elsewhere in Acrobat (Destroy, Mouse Up, Mouse Down, Mouse Enter, Mouse Exit, Page Open, Page Close, Page Visible, Page Invisible, Focus, and Blur). Please see the Events section of the main Acrobat JavaScript documentation for details on those events.

afterBlur

6.0				

The Blur event fires when a MediaPlayer or ScreenAnnot loses the keyboard focus after having it.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onBlur and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

Example

The following script is executed as a Rendition action. The user clicks on the ScreenAnnot to open, but not play the movie clip. Clicking outside the ScreenAnnot (a Blur event) plays the movie. Clicking on the ScreenAnnot (a Focus event) while movie is playing pauses the movie. To continue, the user clicks outside the ScreenAnnot again.

```
var playerEvents = new app.media.Events
({
     afterBlur: function () { player.play(); },
     afterFocus: function () { player.pause(); }
});
var settings = { autoPlay: false };
var args = { settings: settings, events: playerEvents};
var player = app.media.openPlayer(args);
```

See also **`afterFocus`**.

afterClose

6.0				

The Close event fires when a MediaPlayer is closed for any reason.

If you want to start another media player from the Close event, be sure to test **`doc.media.`**canPlay first to make sure playback is allowed. For example, playback may not be allowed because the document is closing.

The Event Object for a Close event includes these properties in addition to the standard Event properties:

`media.closeReason`	Why the player was closed, from **`app.media.closeReason`**
`media.hadFocus`	Did the player have the focus when it was closed?

When a player closes while it has the focus, it first receives a Blur event and then the Close event. In the Close event, **`media.hadFocus`** indicates whether the player had the focus before closing.

When the **`afterClose`** event method is called, the MediaPlayer has already been deleted and its JavaScript object is dead.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onClose and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

Example

See onClose for a representative example.

afterDestroy

6.0				

The Destroy event fires when a ScreenAnnot is destroyed.

When the afterDestroy event method is called, the ScreenAnnot has already been deleted from the document and its JavaScript object is dead.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onDestroy and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterDone

6.0				

The Done event fires when media playback reaches the end of media.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onDone and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterError

6.0				

The Error event fires when an error occurs in a MediaPlayer.

The Event object for an Error event includes these properties in addition to the standard Event properties:

`media.code`	Status code value
`media.serious`	True for serious errors, false for warnings
`media.text`	Error message text

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onError and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterEscape

6.0				

The Escape event fires when the user presses the Escape key while a MediaPlayer is open and has the keyboard focus. A MediaPlayer may receive an Escape event before it receives the Ready event.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onEscape and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterEveryEvent

6.0				

If an Events Object contains an **`onEveryEvent`** or **`afterEveryEvent`** property, its event listener function(s) are called for every event, not just a specific one.

The event listener function(s) in an onEveryEvent or afterEveryEvent property are called before any listener functions that name the specific event.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onEveryEvent and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

Example:

```
var events = new app.media.Events(
{
     // This is called immediately during every event:
     onEveryEvent: function( e )
     { console.println( 'onEveryEvent, event = ' + e.name ); },

     // This is called during a Play event, after onEveryEvent is
     // called:
     onPlay: function() { console.println( "onPlay" ); },

     // This is called for every event, but later during idle time:
     afterEveryEvent: function( e )
     { console.println( "afterEveryEvent, event = " + e.name ); },

     // This is called during idle time after a Play event,
     // and after afterEveryEvent is called:
     afterPlay: function() { console.println( "afterPlay" ); },
});
```

afterFocus

6.0				

The Focus event fires when a MediaPlayer or ScreenAnnot gets the keyboard focus.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onFocus and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

Example

See `afterBlur` for an example of usage.

afterPause

6.0				

The Pause event fires when media playback pauses, either because of user interaction or when the `pause()` method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onPause and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterPlay

6.0				

The Play event fires when media playback starts or resumes, either because of user interaction or when the `play()` method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onPlay and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

afterReady

6.0				

The Ready event fires when a newly-created MediaPlayer is ready for use. Most methods of a MediaPlayer Object cannot be called until the Ready event fires.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onReady and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

See **`afterScript`**, below, **`Markers.get`** and the MediaOffset Object.

Example

This (document level) script plays multiple media clips. For each ScreenAnnot, a media (OpenPlayer) player is opened. When it is ready, the **`afterReady`** script signals this fact to Multiplayer.

```
// Parameters: doc, page, rendition/annot name, mulitPlayer instance
function OnePlayer( doc, page, name, multiPlayer )
{
     var player = app.media.openPlayer({
            annot: doc.media.getAnnot(
              { nPage: page, cAnnotTitle: name }),
            rendition: doc.media.getRendition( name ),
            settings: { autoPlay: false },
            events: {
                   afterReady: function( e ) {
                   multiPlayer.afterReady( player );
                   },
            }
    });
    return player;
}
// Parameters: doc, page, list of rendition/annot names
function MultiPlayer( doc, page )
```

```
{
     var nPlayersCueing = 0;  // number of players cueing up
     var players = [];          // the SinglePlayers

     this.afterReady = function( player ) {
            if( ! player.didAfterReady ) {
                   player.didAfterReady = true;
                   nPlayersCueing--;
                   if( nPlayersCueing == 0 ) this.play();
            }
    }
    this.play = function() {
        for( var i = 0;  i < players.length;  i++ ) players[i].play();
     }
    for( var i = 2;  i < arguments.length;  i++ ) {
         players[i-2] = new OnePlayer(doc,page,arguments[i],this );
         nPlayersCueing++;
    }
}
```

Playing multiple media clips is accomplished by executing the code

```
var myMultiPlayer = new MultiPlayer( this, 0, "Clip1", "Clip2" );
```

from, for example, a mouse up action of a form button.

See **`afterScript`** for another example of **`afterReady`**.

afterScript

6.0				

The Script event fires when a script trigger is encountered in the media during playback.

The Event Object for a Script event includes these properties in addition to the standard Event properties:

`media.command`	Command name
`media.param`	Command parameter string

These two strings can contain any values that the media clip provides. They do not necessarily contain executable JavaScript code it is up to the onScript or afterScript event listener to interpet them.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onScript and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

Example

The following is part of a complete example presented after **`MediaPlayer.seek()`**. The media is an audio clip (`.`wma), which does support markers and scripts, of (famous) quotations. The **`afterReady`** listener counts the number of markers, one at the beginning of each quotation. At the end of each quotation, there is also a embedded command script, the **`afterScript`** listener watches for these commands, and if it is a “pause” command, it pauses the player.

```
var nMarkers=0;
var events = new app.media.Events
events.add({
    // count the number of quotes in this audio clip, save as nMarkers
    afterReady: function() {
        var g = player.markers;
        while ( (index =  g.get( { index: nMarkers } ) ) != null )
           nMarkers++;
    },
    // Each quote should be followed by a script, if the command is to
    // pause, then pause the player.
    afterScript: function( e ) {
        if ( e.media.command == "pause" ) player.pause();
    }
});
var player = app.media.openPlayer({
     rendition: this.media.getRendition( "myQuotes" ),
     settings: { autoPlay: false },
     events: events
});
```

afterSeek

6.0				

The Seek event fires when a MediaPlayer is finished seeking to a playback offset as a result of a **`seek()`** call. Note that not all media players fire Seek events.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onSeek and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

afterStatus

6.0				

The Status event fires on various changes of status that a MediaPlayer reports.

The Event Object for a Status event includes these properties in addition to the standard Event properties:

`media.code`	Status code value, defined in `app.media.status`
`media.text`	Status message text

The following values are used only by some media players, and only when `media.code == app.media.status.buffering`. They are zero otherwise.

`media.progress`	Progress value from 0 to media.total
`media.total`	Maximum progress value

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onStatus and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

Example

The following code would monitor the status of the player, as executed from a Rendition event associated with a ScreenAnnot.

```
var events = new app.media.Events
```

```
events.add({
     afterStatus: function ( e ) {
        console.println( "Status code " + e.media.code +
           ", description: " + e.media.text);
     }
});
app.media.openPlayer({ events: events });
```

afterStop

6.0				

The Stop event fires when media playback stops, either because of user interaction or when the **`stop()`** method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the onStop and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onBlur

6.0				

The Blur event fires when a MediaPlayer or ScreenAnnot loses the keyboard focus after having it.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterBlur and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onClose

6.0				

The Close event fires when a MediaPlayer is closed for any reason.

If you want to start another media player from the Close event, be sure to test **`doc.media.canPlay`** first to make sure playback is allowed. For example, playback may not be allowed because the document is closing.

The Event object for a Close event includes these properties in addition to the standard Event properties:

`media.closeReason`	Why the player was closed, from **`app.media.closeReason`**
`media.hadFocus`	Did the player have the focus when it was closed?

When a player closes while it has the focus, it first receives a Blur event and then the Close event. In the Close event, media.hadFocus indicates whether the player had the the focus before closing.

When the afterClose event method is called, the MediaPlayer has already been deleted and its JavaScript object is dead.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterClose and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

Example

This script gets information about why the media clip closed, executed from a Rendition action. See **`app.media.closeReason`**,

```
var playerEvents = new app.media.Events({
     onClose: function (e) {
            var eReason, r = app.media.closeReason;
            switch ( e.media.closeReason )
            {
                 case r.general: eReason = "general"; break;
                 case r.error: eReason = "error"; break;
                 case r.done: eReason = "done"; break;
                 case r.stop: eReason = "stop"; break;
                 case r.play: eReason = "play"; break;
                 case r.uiGeneral: eReason = "uiGeneral"; break;
```

```
                    case r.uiScreen: eReason = "uiScreen"; break;
                    case r.uiEdit: eReason = "uiEdit"; break;
                    case r.docClose: eReason = "Close"; break;
                    case r.docSave: eReason = "docSave"; break;
                    case r.docChange: eReason = "docChange"; break;
               }
               console.println("Closing...The reason is  " + eReason );
        }
  });
  app.media.openPlayer({ events: playerEvents })
```

onDestroy

6.0				

The Destroy event fires when a ScreenAnnot is destroyed.

Parameters

oMediaEvent	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterDestroy and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onDone

6.0				

The Done event fires when media playback reaches the end of media.

Parameters

oMediaEvent	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterDone and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onError

6.0				

The Error event fires when an error occurs in a MediaPlayer.

The Event object for an Error event includes these properties in addition to the standard Event properties:

`media.code`	Status code value
`media.serious`	`true` for serious errors, `false` for warnings
`media.text`	Error message text

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterError and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onEscape

6.0				

The Escape event fires when the user presses the Escape key while a MediaPlayer is open and has the keyboard focus. A MediaPlayer may receive an Escape event before it receives the Ready event.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterEscape and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onEveryEvent

6.0				

If an Events Object contains an onEveryEvent or afterEveryEvent property, its event listener function(s) are called for every event, not just a specific one.

The event listener function(s) in an onEveryEvent or afterEveryEvent property are called before any listener functions that name the specific event.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterEveryEvent and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

onFocus

6.0				

The Focus event fires when a MediaPlayer or ScreenAnnot gets the keyboard focus.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterFocus and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

onGetRect

6.0				

The GetRect event fires whenever the multimedia plug-in needs to get the display rectangle for a docked MediaPlayer.

The Event object for an GetRect event includes this property in addition to the standard Event properties:

`media.rect`	Player rectangle, an array of four numbers in device space

The **`onGetRect()`** method must set this property in the **`oMediaEvent`** before returning.

Although you can write an **`afterGetRect`** listener, there is no useful purpose for it if it returns a rect property it will be ignored. The **`onGetRect`** listener is where the rect property must be set.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

Example

Page 0 has a series of (thumbnail-size) ScreenAnnots, and page 1 is a blank page. Put the viewer into continuous facing mode so that both pages are seen side-by-side. Below is a typical Rendition action or mouse up button JavaScript action.

```
var rendition = this.media.getRendition("Clip1");
var settings = rendition.getPlaySettings();
var annot = this.media.getAnnot({ nPage:0,cAnnotTitle:"ScreenClip1" });
var player = app.media.openPlayer({
     rendition: rendition,
     annot: annot,
     settings: { windowType: app.media.windowType.docked },
     events:
     {
            onGetRect: function (e) {
                   var width = e.media.rect[2] - e.media.rect[0];
                   var height = e.media.rect[3] - e.media.rect[1];
                   width *= 3; // triple width and height
                   height *= 3;
                   e.media.rect[0] = 36; // move left,upper to
                   e.media.rect[1] = 36; // upper left-hand corner
                   e.media.rect[2] = e.media.rect[0]+width;
                   e.media.rect[3] = e.media.rect[1]+height;
                   return e.media.rect; // return this
            }
     }
});
player.page = 1; // show on page 1, this triggers an onGetRect event.
```

See **`MediaPlayer.page`** and **`MediaPlayer.triggerGetRect`** for a variation on this same example.

onPause

6.0				

The Pause event fires when media playback pauses, either because of user interaction or when the **`play()`** method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterPause and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onPlay

6.0				

The Play event fires when media playback starts or resumes, either because of user interaction or when the **`pause()`** method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterPlay and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onReady

6.0				

The Ready event fires when a newly-created MediaPlayer is ready for use. Most methods of a MediaPlayer Object cannot be called until the Ready event fires.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterReady and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onScript

6.0				

The Script event fires when a script trigger is encountered in the media during playback.

The Event object for a Script event includes these properties in addition to the standard Event properties:

`media.command`	Command name
`media.param`	Command parameter string

These two strings can contain any values that the media clip provides. They do not necessarily contain executable JavaScript code it is up to the onScript or afterScript event listener to interpet them.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterScript and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onSeek

6.0				

The Seek event fires when a MediaPlayer is finished seeking to a playback offset as a result of a `seek()` call. Note that not all media players fire Seek events.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterSeek and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onStatus

6.0				

The Status event fires on various changes of status that a MediaPlayer reports.

The Event Object for a Status event includes these properties in addition to the standard Event properties:

`media.code`	Status code value, defined in `app.media.status`
`media.text`	Status message text

The following values are used only by some media players, and only when `media.code == app.media.status.buffering`. They are zero otherwise.

`media.progress`	Progress value from 0 to media.total
`media.total`	Maximum progress value

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterStatus and the explanation of the differences between an “on” event and an “after” event in EventListener Object.

onStop

6.0				

The Stop event fires when media playback stops, either because of user interaction or when the `stop()` method is called.

Parameters

`oMediaEvent`	An Event Object which is automatically passed to this event listener.

Returns

Nothing

See the afterStop and the explanation of the differences between an "on" event and an "after" event in EventListener Object.

Marker Object

A Marker object represents a named location in a media clip that identifies a particular time or frame number, similar to a track on an audio CD or a chapter on a DVD. Markers are defined by the media clip itself.

A Marker object can be obtained from the `Markers.get()` method.

Marker Object Properties

frame

6.0				

A frame number, where 0 represents the beginning of media. For most players, markers have either a frame or a time value, but not both.

Type: Number *Access: R.*

index

6.0				

An arbitrary index number assigned to this marker. Markers have sequential index numbers beginning with 0, but these index numbers may not be in the same order that the markers appear in the media.

Type: Number	*Access: R.*

name

6.0				

The name of this marker. Each marker in a media clip has a unique name.

Type: String	*Access: R.*

Example

Get a marker by its index, then print the name of the marker to the console.

```
// assume player is a MediaPlayer object
var markers = player.markers;
// get marker with index of 2
var markers = g.get( { index: 2 } );
console.println( "The marker with index of " + markers.index
     +", has a name of " + index.name );
```

time

6.0				

A time in seconds, where 0 represents the beginning of media. For most players, markers have either a frame or a time value, but not both.

Type: Number	*Access: R.*

Example

Get a named marker, then print the time in seconds from the beginning of the media, of that marker.

```
// assume player is a MediaPlayer object
var markers = player.markers;
// get marker with name of "Chapter 1"
var markers = g.get( { name: "Chapter 1" } );
console.println( "The named marker \"Chapter 1\”, occurs at time "
     + markers.time);
```

Markers Object

The markers property of a MediaPlayer is a Markers object which represents all of the markers found in the media clip currently loaded into the player. A marker is a named location in a media clip that identifies a particular time or frame number, similar to a track on an audio CD or a chapter on a DVD. Markers are defined by the media clip itself.

The constructor is **`app.media.Markers`**.

Markers Object Properties

player

6.0				

The MediaPlayer Object that this Markers object belongs to, i.e.

```
player.markers.player == player.
```

Type: MediaPlayer Object   Access: R.

Markers Object Methods

get

6.0				

The **`Markers.get()`** method looks up a marker by name, index number, time in seconds, or frame number, and returns the Marker Object representing the requested marker. The object parameter should contain either a name, index, time, or frame property. A marker name can also be passed in directly as a string.

If a time or frame is passed in, the nearest marker at or before that time or frame is returned. If the time or frame is before any markers in the media, then null is returned.

Parameters

An object or string representing the name, index number, time in seconds, or the frame number of the marker. The object parameter should contain either a name, index, time, or frame property. A marker name can also be passed in directly as a string.

Returns

Marker Object or **`null`**

Marker index numbers are assigned sequentially starting with 0, and they are not necessarily in order by time or frame. In particular, note that these are not the same values that Windows Media Player uses for marker numbers. To find all of the available markers in a media clip, call **`MediaPlayer.markers.get()`** in a loop starting with **`{index: 0}`** and incrementing the number until **`get()`** returns **`null`**.

Example:

This example counts the number of markers on the media clip.

```
var index, i =  0;
// assume player is a MediaPlayer object.
var m = player.markers;
while ( (index = m.get( { index: i } ) ) != null ) i++;
console.println("There are " + i + " markers.");
```

Example:

```
// Get a marker by name, two different ways
var marker = player.markers.get( "My Marker" );
var marker = player.markers.get({ name: "My Marker" });
// Get a marker by index
var marker = player.markers.get({ index: 1 });
// Get a marker by time
var marker = player.markers.get({ time: 17.5 });
// Get a marker by frame
var marker = player.markers.get({ frame: 43 });
```

MediaOffset Object

A MediaOffset represents a position in a MediaClip, either in terms of time or a frame count.

This position can either be relative to a named marker, or it can be an absolute position (i.e. relative to the beginning of the media).

The MediaOffset can be specified either as an object with the properties named below, or it can simply be a number, which is interpreted as **`{time: number}`**.

Some media formats (e.g. QuickTime) are time-based and others (e.g. Flash) are frame-based. A MediaOffset that specifies a time or frame must match the media format in use. If both time and frame are specified, the results are undefined: the incorrect one may be ignored, or a JavaScript exception may be thrown.

The MediaOffset object is used by **`MediaPlayer.seek()`**, **`MediaPlayer.where()`**, **`MediaSettings.endAt`** and **`MediaSettings.startAt`**.

MediaOffset Object Properties

frame

6.0				

A frame number. If the **`marker`** property is also present, this frame number is relative to the specified marker and may be positive, negative, or zero. Otherwise, it is relative to the beginning of media and may not be negative. Note that **`{frame: 0}`** represents the beginning of media.

Type: Number *Access: R/W.*

marker

6.0				

The name of a specific marker in the media.

Type: String *Access: R/W.*

time

6.0				

A time in seconds, or **`Infinity`**. If the **`marker`** property is also present, this time is relative to the specified marker and is a nonnegative value, but not **`Infinity`**. Otherwise, the time is relative to the beginning of media and must not be negative. Note that the offset { time: 0 } represents the beginning of media.

Type: Number *Access: R/W.*

Example

Below are examples of absolute and relative offsets

```
{ time: 5.4 } // offset 5.4 seconds from the beginning of media
{ marker: "Chapter 1", time: 17 } // 17 seconds after "Chapter 1"
```

These offsets can be used by the **`MediaPlayer.seek()`** method:

```
// assume player is a MediaPlayer object
player.seek({ time: 5.4 });
player.seek({ marker: "Chapter 1", time: 17 });
```

MediaPlayer Object

A MediaPlayer object represents an instance of a multimedia player such as QuickTime, Windows Media Player, or others. Its **`settings`** and **`events`** properties let you manipulate the player from JavaScript code and handle events that the player fires. MediaPlayer is not part of a PDF file; it is a transient object created in memory when needed.

MediaPlayer Object Properties

annot

6.0				

`MediaPlayer.annot` is a reference to the ScreenAnnot associated with a MediaPlayer. This property exists only for a MediaPlayer object that is connected to a ScreenAnnot. The property is set by **`app.media.addStockEvents()`** or by methods that call **`addStockEvents()`** indirectly, such as **`app.media.openPlayer()`**.

Type: ScreenAnnot Object Access: R/W.

defaultSize

6.0				

The **`MediaPlayer.defaultSize`** property is a read-only object containing the width and height of the MediaPlayer's MediaClip:

```
{ width: number, height: number }
```

If the media player is unable to provide this value, then defaultSize is **`undefined`**.

Type: Object Access: R.

doc

6.0				

`MediaPlayer.doc` is a reference to the Doc object that owns the MediaPlayer.

Type: Object Access: R.

events

6.0				

The **`MediaPlayer.events`** property is a Events Object containing the event listeners that are attached to a MediaPlayer. See Events Object for details.

Type: Events Object *Access: R/W.*

Example

Create a media player, then modify the events of that player. The script is executed as a Rendition action with an associated rendition.

```
var events = new app.media.Events;
var player = app.media.createPlayer();
player.events.add({
     onReady: function() { console.println("The player is ready"); }
});
player.open();
```

hasFocus

6.0				

The **`MediaPlayer.hasFocus`** property is **`true`** if the media player is open and has the keyboard focus.

Type: Boolean *Access: R.*

id

6.0				

The **`MediaPlayer.id`** property contains the player ID for the player software that this player is using. It is **`undefined`** if the player has not been opened. This player ID is the same value that is found in **`PlayerInfo.id`** for the media player software that implements this player.

Type: Boolean *Access: R.*

Example

Print player id to the console

```
// assume args has been defined
var player = app.media.openPlayer( args )
console.println("player.id = " + player.id);
// in the console, this script could possibly print...
player.id = vnd.adobe.swname:ADBE_MCI
```

innerRect

6.0				

The **`MediaPlayer.innerRect`** property is a rectangle array representing the player's inner rectangle on the screen. As with other such arrays in Acrobat JavaScript, the coordinates are in the order [left, top, right, bottom]. The rectangle does *not* include any window title, or other such gadgets around the edges of the player, but it does include the player controller if a controller is present. It is undefined if the player is not open.

For a docked media player, this rectangle is in device space and is read-only: It will throw an exception if you try to set it. Instead, use **`MediaPlayer.triggerGetRect()`** to cause a docked player to be resized. For a floating media player, the rectangle is in screen coordinates and is writable, but the user's security settings may override a value you set here. For example, if you try to move a floating media player offscreen, it may be forced back onscreen. This will not throw an exception. You can read this property after writing it to see if your value was overridden.

Type: Array *Access: R or R/W.*

See also, outerRect.

isOpen

6.0				

`MediaPlayer.isOpen`, a boolean, is **`true`** if the media player is currently open. Use **`MediaPlayer.open()`** and **`MediaPlayer.close()`** to open or close a player.

Type: Boolean *Access: R.*

isPlaying

6.0				

The **`MediaPlayer.isPlaying`** property is **`true`** if the media is currently playing. It is **`false`** if the player is not open, or if the media is paused, stopped, fast forwarding or rewinding, or in any other state.

Type: Boolean *Access: R.*

markers

6.0				

`MediaPlayer.markers` is a collection of all the markers available for the current media.

See Markers Object for details of this property.

Type: Markers Object   *Access: R.*

Example

See Example 2 following **`MediaPlayer.seek()`** for an illustration of usage.

outerRect

6.0				

`MediaPlayer.outerRect` is a rectangle Array representing the player's outer rectangle on the screen. As with other such arrays in Acrobat JavaScript, the coordinates are in the order [left, top, right, bottom]. This rectangle includes any player controller, window title, and other such gadgets around the edges of the player. It is **`undefined`** if the player is not open.

For a docked media player, this rectangle is in device space and is read-only: It will throw an exception if you try to set it. Instead, use **`MediaPlayer.triggerGetRect()`** to cause a docked player to be resized. For a floating media player, the rectangle is in screen coordinates and is writable, but the user's security settings may override a value you set here. For example, if you try to move a floating media player offscreen, it may be forced back onscreen. This will not throw an exception. You can read this property after writing it to see if your value was overridden.

Type: Array   *Access: R or R/W.*

See also innerRect.

page

6.0				

`MediaPlayer.page` is the page number in which a docked media player appears. It is **`undefined`** for players that are not docked. A docked media player can be moved to another page by changing its **`page`** property, and this triggers a GetRect (see onGetRect) event.

Type: Number   *Access: R/W.*

Example

Play a media clip on page 1 (base zero). The placement of the media player on page 1 is the same the ScreenAnnot on page 0.

```
var player = app.media.openPlayer({
        rendition: this.media.getRendition( "myClip" ),
        annot: this.media.getAnnot({ nPage:0, cAnnotTitle:"myScreen" }),
        settings: { windowType: app.media.windowType.docked }
```

```
    });
player.page = 1;
```

See **`onGetRect`** and **`triggerGetRect`** for variations on this same example.

settings

6.0				

`MediaPlayer.settings` includes all of the settings that are used to create a MediaPlayer. See MediaSettings Object for a complete list.

NOTE: In Acrobat 6.0, changing a property in **`MediaPlayer.settings`** after the player has been created has no effect. This may be changed in a future release to make these settings live. For compatibility with both current and future releases, avoid changing any settings properties while a player is open.

Type: MediaSettings Object Access: R/W.

uiSize

6.0				

`MediaPlayer.uiSize` is an array containing the size of the controller of the player for each edge of the player, in the same order as a window rectangle: [left, top, right, bottom]. Each of these values is normally a positive value or zero. These values do not include window gadgets such as title bars, only the controller.

This property is not available until the Ready event is fired (see onReady and afterReady). Unlike most MediaPlayer properties, it is permissible to read it during an “on” event method such as onReady.

Type: Array *Access: R.*

Example

Get the **`uiSize`** of the player. This code is executed as a Rendition action event.

```
var args = {
     events: {
          onReady: function () {
               console.println("uiSize = " + player.uiSize );
          }
     }
};
var player = app.media.openPlayer(args);
```

visible

6.0				

The **`MediaPlayer.visible`** property controls whether the player is visible. Unlike **`MediaPlayer.settings.visible`**, this property takes effect immediately. If the player is not open, reading this property returns **`undefined`** and setting it throws an exception.

Setting this property may fire events. For example, if the player is visible and has the focus, making it invisible fires a Blur event.

Type: Boolean *Access: R/W.*

Example

Play the audio *only* of a video clip

```
// assume a definition of args
var player = app.media.openPlayer(args);
player.visible = false;
```

MediaPlayer Object Methods

close

6.0				

Closes the media player if it is open. Does nothing (and is not an error) if the player is closed.

The **`eReason`** parameter should be a value from the **`app.media.closeReason`** enumeration. This value is passed through to the **`event.media.closeReason`** property for the Close event (see **`onClose`** and **`afterClose`**) that the **`close()`** method fires.

If the player has the keyboard focus, a Blur event (**`onBlur`**/**`afterBlur`**) is fired before the Close event. Other events, such as Status (**`onStatus`**/**`afterStatus`**) and Stop (**`onStop`**/**`afterStop`**), may also be fired depending on the particular media player.

Parameters

`eReason`	**`eReason`** is a value from the **`app.media.closeReason`** enumeration.

Returns

Nothing

open

6.0				

The **`MediaPlayer.open()`** method attempts to open the media player as specified by **`MediaPlayer.settings`**. If the player is already open, an exception is thrown. If the player was previously opened and then closed, **`open()`** may be called to open the player again. This uses the same JavaScript object as before but opens a new instance of the actual media player (e.g. the new player does not remember the playback position from the old player).

For a docked player, a GetRect event (**`onGetRect`**) is fired when the player is opened.

If **`MediaPlayer.settings.autoPlay`** is **`true`** (which it is by default), then playback begins and a Play event (**`onPlay`**/**`afterPlay`**) is fired.

The **`open()`** method may result in a security prompt dialog depending on the user's settings. This may also result in events being fired to other media players, screen annots, or other objects. For example, if another media player has the keyboard focus, it will receive a Blur event (**`onBlur`**/**`afterBlur`**).

If **`bAllowSecurityUI`** is **`false`**, then **`open()`** never displays a security prompt, but returns a failure code instead.

For a media player in a floating window, additional security checks are made against the user's settings. For example, the user may specify that title bars are required on all floating player windows. If **`MediaPlayer.settings.floating`** contains options that the user does not allow, then **`bAllowFloatOptionsFallback`** controls what happens. If it is **`false`**, playback is disallowed and an error code is returned. If it is **`true`**, then the options in **`MediaPlayer.settings.floating`** are changed as needed to conform to the user's security settings, and then **`open()`** proceeds with those changed settings.

The return value is an object which currently contains one property, **`code`**, which is a result code from the **`app.media.openCode`** enumeration. If your PDF is opened in a future version of Acrobat, there may be additional properties in this object, or a code value added in that future version. Be sure to handle any such values gracefully.

Parameters

`bAllowSecurityUI`	(optional) The default is **`true`**. See the description of this parameter given above.
`bAllowFloatOptionsFallback`	(optional) The default is **`true`**. See the description of this parameter given above.

Returns

An object with a **`code`** property

Example

See "Example 1" on page 30 for an example of usage.

pause

6.0				

Pauses playback of the current media and fires a Pause event (**`onPause`**/**`afterPause`**). The Pause event may occur during the **`pause()`** call or afterward, depending on the player.

The **`pause()`** method has no effect if the media is already paused or stopped, or if playback has not yet started or has completed. Not every media player supports **`pause()`**, and not every media format supports it; in particular, most streaming formats do not support **`pause()`**. Players may either throw an exception or silently ignore **`pause()`** in these cases.

Parameters

None

Returns

Nothing

Example

See Example 2 following the **`seek()`** method below for an example of usage.

play

6.0				

Starts playback of the current media and fires a Play event (**`onPlay`**/**`afterPlay`**). The Play event may occur during the **`play()`** call or afterward, depending on the player.

If the media is already playing, it continues playing and no event is fired. If it is paused, rewinding, or fast forwarding, it resumes playback at the current position. If it is stopped, either at the beginning or end of media, playback starts from the beginning.

Parameters

None

Returns

Nothing

Example

See Example 2 following the **`seek()`** method below for an example of usage.

seek

6.0				

Sets the current media's playback location to the position described by the MediaOffset Object contained in **`oMediaOffset`**.

If the media is playing, it continues playing at the new location. If the media is paused, it moves to the new location and remains paused there. If the media is stopped, the result will vary depending on the player.

Different media players handle seek errors in different ways: Some ignore the error and others throw a JavaScript exception.

Most, but not all, media players fire a Seek event (**`onSeek`**/**`afterSeek`**) when a seek is completed.

The seek operation may take place during the execution of the **`seek()`** method or later, depending on the player. If **`seek()`** returns before the seek operation is completed and you call another player method before the seek is completed, the results will vary depending on the player.

Parameters

`oMediaOffset`	A MediaOffset Object, the properties of which indicate the playback location to be set.

Returns

Nothing

Example 1

```
// Rewind the media clip
player.seek({ time: 0 });

// Play starting from marker "First"
player.seek({ marker: "First" });

// Play starting five seconds after marker "One"
player.seek({ marker: "One", time: 5 });
```

Example 2

The following script randomly plays (famous) quotations. The media is an audio clip (`.`wma), which does support markers and scripts, of (famous) quotations. The **`afterReady`** listener counts the number of markers, one at the beginning of each quotation. At the end of each quotation, there is also a embedded command script, the **`afterScript`** listener watches for these commands, and if it is a “pause” command, it pauses the player.

```
var nMarkers=0;
var events = new app.media.Events
```

```
events.add({
    // count the number of quotes in this audio clip, save as nMarkers
     afterReady: function()
     {
            var g = player.markers;
            while ( (index =  g.get( { index: nMarkers } ) ) != null )
            nMarkers++;
     },
     // Each quote should be followed by a script, if the command is to
     //     pause, then pause the player.
     afterScript: function( e ) {
        if ( e.media.command == "pause" ) player.pause();
     }
});
var player = app.media.openPlayer({
     rendition: this.media.getRendition( "myQuotes" ),
     settings: { autoPlay: false },
     events: events
});
// randomly choose a quotation
function randomQuote() {
     var randomMarker, randomMarkerName;
     console.println("nMarkers = " + nMarkers);
     // randomly choose an integer between 1 and nMarkers, inclusive
     randomMarker = Math.floor(Math.random() * 100) % ( nMarkers ) + 1;
     // indicate what quotation we are playing
     this.getField("Quote").value = "Playing quote " + randomMarker;
     // The marker names are "quote 1", "quote 2", "quote 3", etc.
     randomMarkerName = "quote " + randomMarker;
     // see the marker with the name randomMarkerName
     player.seek( { marker: randomMarkerName } );
     player.play();
}
```

Action is initiated by the mouse up button action such as

```
try { randomQuote() } catch(e) {}
```

setFocus

6.0				

Sets the keyboard focus to the media player and fires a Focus event (**onFocus**/**afterFocus**). If another player or PDF object has the focus, that object receives a Blur event (**onBlur**/**afterBlur**). If the media player already has the focus, nothing happens. If the player is not open or not visible, an exception is thrown.

Parameters

None

Returns

Nothing

Example

See “Example 1” on page 30 for an example of usage.

stop

6.0				

Stops playback of the current media, if it is playing or paused, and fires a Stop event (**`onStop`**/**`afterStop`**). The Stop event may occur during execution of the **`stop()`** method or afterward, depending on the player. Does nothing if the media is not playing or paused.

Throws an exception if the player is not open.

After playback stops, the player sets the media position to either the beginning or end of media, depending on the player. If **`MediaPlayer.play()`** is called after this, playback starts at the beginning of media.

Parameters

None

Returns

Nothing

triggerGetRect

6.0				

Fires a GetRect event (see **`onGetRect`**) to cause a docked media player to be resized.

Parameters

None

Returns

Nothing

Example

This example is similar to the one that follows **`onGetRect`**. Page 0 has a series of (thumbnail-size) ScreenAnnots. Below is a typical Rendition action or mouse up button JavaScript action, when the action is executed, the media clip is resized and played.

```
var rendition = this.media.getRendition("Clip1");
var annot = this.media.getAnnot({ nPage:0,cAnnotTitle:"ScreenClip1" });
var player = app.media.openPlayer({
     rendition: rendition,
     annot: annot,
     settings: { windowType: app.media.windowType.docked },
     events: {
             onGetRect: function (e) {
                  var width = e.media.rect[2] - e.media.rect[0];
                  var height = e.media.rect[3] - e.media.rect[1];
                  width *= 3; // triple width and height
                  height *= 3;
                  e.media.rect[0] = 36; // move left,upper to
                  e.media.rect[1] = 36; // upper left-hand corner
                  e.media.rect[2] = e.media.rect[0]+width;
                  e.media.rect[3] = e.media.rect[1]+height;
                  return e.media.rect; // return this
             }
     }
});
player.triggerGetRect(); // trigger the onGetRec event
```

where

6.0				

Reports the current media's playback location in a MediaOffset Object. This object contains either a time or frame property, depending on the media player and media type.

Throws an exception if the player is not open or if the player does not support **`where()`**.

Parameters

None

Returns

MediaOffset Object

Example:

```
// What is the playback location in seconds?
// This code assumes that the player supports where() using time.
var where = player.where();
var seconds = where.time;
// What chapter (marker) are we in?
var marker = player.markers.get({ time: seconds });
var name = marker ? marker.name : "no marker";
```

MediaReject Object

A MediaReject provides information about a Rendition that was rejected by a **`Rendition.select()`** call. It includes a reference to the original Rendition along with the reason why it was rejected. In a MediaSelection Object returned by **`select()`**, **`MediaSelection.rejects`** is an array of MediaReject objects.

MediaReject Object Properties

rendition

6.0				

`MediaSelection.rendition` is a reference to the Rendition that was rejected in a **`select()`** call.

Type: Rendition Object   *Access: R.*

Example

Get a list of rejected renditions. The script is executed as a Rendition action.

```
selection = event.action.rendition.select(true);
for ( var i=0; i<selection.rejects.length; i++)
     console.println("Rejected Renditions: "
            + selection.rejects[i].rendition.uiName);

// now play the first available rendition.
console.println( "Preparing to play " + selection.rendition.uiName);
var settings = selection.rendition.getPlaySettings();
var args = {
    rendition: selection.rendition,
    annot: this.media.getAnnot({ nPage: 0, cAnnotTitle: "myScreen" }),
    settings: settings
};
app.media.openPlayer(args);
```

MediaSelection Object

`Rendition.select()` returns a MediaSelection, an object which can then be used to create a MediaSettings Object for playback.

MediaSelection Object Properties

selectContext

6.0				

`MediaSelection.selectContext` is a value that can be used to write a loop that calls **`Rendition.select()`** repeatedly to do a customized selection based on any criteria that you can test in JavaScript code.

Type: Object *Access: R.*

Example:

```
function MyTestSelection( selection )
{
     // This function should test the selection as you wish and return
     // true to use it or false to reject it and try another one.
}
function MyGetSelection( rendition )
{
     var selection;
     for( selection = rendition.select(); selection;
            selection = rendition.select
                   ({ oContext: selection.selectContext }))
     {
            if( MyTestSelection( selection ) )
                   break;
     }
     return selection;
}
```

players

6.0				

`MediaSelection.players` is an array of strings identifying the media players that may be used to play **`MediaSelection.rendition`**. Both the players and rendition properties are **`null`** if no playable rendition is found.

Type: Array of String *Access: R.*

Example

Get a list of the players that will play the selected rendition. The code below assumes execution as a Rendition action.

```
var selection = event.action.rendition.select();
for ( var o in selection.players )
```

```
            console.println( selection.players[o].id );
```

rejects

6.0				

MediaSelection.rejects is an array of MediaReject Objects. These are the Renditions that were rejected by the **Rendition.select()** call that returned this MediaSelection. See MediaReject Object for details.

Type: Array of MediaReject Objects *Access: R.*

Example

See the Example following **MediaReject.rendition** for an example.

rendition

6.0				

MediaSelection.rendition is the selected rendition, or **null** if none was playable.

Type: Rendition Object *Access: R.*

Example

Get the name of the selected rendition. This script is executed from a Rendition action event.

```
var selection = event.action.rendition.select();
console.println( "Preparing to play " + selection.rendition.uiName);
```

MediaSettings Object

A MediaSettings object, which appears in a **MediaPlayer.settings** property, contains settings required to create and open a MediaPlayer. Many of these settings have default values, but some are required depending on the type of player being opened and depending on other settings. See the notes for each MediaSettings property for details.

Acrobat and the various media players will attempt to use these settings, but there is no guarantee that they will all be honored. For example, very few players honor **MediaSettings.palindrome**.

MediaSettings Object Properties

autoPlay

6.0				

The **`MediaSettings.autoPlay`** property specifies whether the media clip should begin playing automatically after the player is opened. If you set **`autoPlay`** to **`false`**, use **`MediaPlayer.play()`** to begin playback. The default value is **`true`**.

Type: Boolean *Access: R/W.*

Example

See the examples following **`afterReady`** and **`players`**.

baseURL

6.0				

`MediaSettings.baseURL` is the base URL to be used to resolve any relative URLs used in the media clip, e.g. if the media opens a web page. There is no default value; if baseURL is not specified, the interpretation of a relative URL will vary depending the media player, but in most cases will not work.

Type: String *Access: R/W.*

bgColor

6.0				

`MediaSettings.bgColor` specifies the background color for the media player window. The array may be in any of the color array formats supported by Acrobat JavaScript.

If bgColor is not specified, the default value depends on the window type:

Docked: White

Floating: The window background color specified in the operating system control panel

Full Screen: The full screen background color specified in the user's Acrobat preferences

Type: Color Array *Access: R/W.*

Example

```
// Red background
settings.bgColor = [ "RGB", 1, 0, 0 ];
```

bgOpacity

6.0				

MediaSettings.bgOpacity specifies the background opacity for the media player window. The value may range from 0.0 (fully transparent) to 1.0 (fully opaque). The default value is 1.0.

Type: Number *Access: R/W.*

endAt

6.0				

MediaSettings.endAt defines the ending time or frame for playback. This may be an absolute time or frame value, or a marker name, or a marker plus a time or frame, as described under MediaOffset Object. Playback ends at the specified time or frame, or as close to that point as the media player is able to stop. If **endAt** is not specified, the default value is the end of media.

See also **startAt**.

Type: MediaOffset Object *Access: R/W.*

Example

The following script plays an audio clip beginning 3 seconds into the media to 8 seconds into the media.

```
var player = app.media.openPlayer({
        rendition: this.media.getRendition( "myAudio" ),
        doc: this,
        settings: {
            startAt: 3,
            endAt: 8
        },
    });
```

data

6.0				

MediaSettings.data, often referred to as the **MediaData object**, is an object that a media player can use to read its media clip data, whether from an external file or embedded in the PDF. The contents of this object are not directly usable from JavaScript.

This data object is obtained in several ways, from **app.media.getAltTextData()**, **app.media.getURLData()**, or indirectly via **Rendition.getPlaySettings()**.

The data object may be bound to the rendition's document, so it may become unavailable if the document is closed.

Type: Object *Access: R.*

Example

See the examples that follow **`app.media.getURLData()`**

duration

6.0				

`MediaSettings.duration` is the amount of time in seconds that playback will take. If not specified, the default is to play the entire media, or the amount of time between the **`startAt`** and **`endAt`** points if either of those is specified.

Note that the duration may be longer than the entire media length or the difference between the **`startAt`** and **`endAt`** points. In that case, playback continues to the end of media or to the **`endAt`** point, and then playback pauses at that location until the duration elapses.

Type: Number *Access: R/W.*

Example

Play a floating window with infinite duration. The "Playback Location" (from the UI) of the rendition is a floating window. The code below is executed from a form button. The floating window will remain open after the player has reached the end of the media. We close the player before opening it again to avoid stacked floating windows.

If this script is executed from a Rendition action, the rendition could be specified through the UI, and closing the player would not be necessary.

```
var rendition = this.media.getRendition("Clip");
if ( player && player.isOpen )
     try { player.close(app.media.closeReason.done); } catch(e) {};
var player = app.media.openPlayer({
    rendition: rendition,
    settings: { duration: Infinity }
});
```

floating

6.0				

`MediaSettings.floating` is an object containing properties that define the location and style of a floating window.

This object is ignored unless **`MediaSettings.windowType`** has a value of **`app.media.windowType.floating`**.

Type: Object *Access: R.*

Properties of floating

Defaults are used for all the floating settings if they are not specified.

Property	Type	Description
`align`	Number	Specifies how the floating window is to be positioned relative to the window specified by the over property. The value of **`align`** is one of the values of `app.media.align`.
`over`	Number	Specifies what window the floating window is to be aligned relative to. The value of **`over`** is one of the values of `app.media.over`.
`canResize`	Number	Specifies whether the floating window may be resized by the user. The value of **`canResize`** is one of the values of `app.media.canResize`.
`hasClose`	Boolean	If **`true`**, the floating window should have a close window control button.
`hasTitle`	Boolean	If **`true`**, a title should be displayed in the title bar.
`title`	String	This title to be displayed if **`hasTitle`** is **`true`**.
`ifOffScreen`	Number	Specifies what action should be taken if the floating window is positioned totally or partially offscreen. The value of ifOffScreen is one of the values of `app.media.ifOffScreen`.
`rect`	Array of four Numbers	An array of screen coordinates specifying the location and size of the floating window. Required if **`width`** and **`height`** are not given.
`width`	Number	The width of the floating window. Required if **`rect`** is not given.
`height`	Number	The height of the floating window. Required if **`rect`** is not given.

Example

```
var rendition = this.media.getRendition( "myClip" );
var floating = {
     align: app.media.align.topCenter,
     over: app.media.over.appWindow,
     canResize: app.media.canResize.no,
     hasClose: true,
     hasTitle: true,
     title: rendition.altText,
     ifOffScreen: app.media.ifOffScreen.forceOnScreen,
     width: 400,
     height: 300
};
var player = app.media.openPlayer({
     rendition: rendition,
     settings: {
         windowType: app.media.windowType.floating,
         floating: floating
     }
});
```

layout

6.0				

`MediaSettings.layout` is a value chosen from the **`app.media.layout`** enumeration, which defines whether and how the content should be resized to fit the window. The default value varies with different media players.

Type: Number *Access: R/W.*

monitor

6.0				

For a full screen media player, **`MediaSettings.monitor`** determines which display monitor will be used for playback. This may be either a Monitor Object or a Monitors Object. If it is an array, the first element (which is a Monitor object) is used.

Type: Monitor or Monitors object *Access: R/W.*

NOTE: Only the **`rect`** property **`MediaSettings.monitor.rect`** (in the case of a Monitor object) or **`MediaSettings.monitor[0].rect`** (for a Monitors object) is used for playback.

See **`monitorType`** (below) for a discussion of the relationship between the **`monitor`** and **`monitorType`** properties.

Example

Play a media clip in full screen from a form button.

```
    var player = app.media.openPlayer({
        rendition: this.media.getRendition("Clip"),
        settings: {
            monitor: app.monitors.primary(),
            windowType: app.media.windowType.fullScreen,
        }
});
```

NOTE: The user trust manager settings must allow fullscreen play back.

monitorType

6.0				

`MediaSettings.monitorType` is an **`app.media.monitorType`** value that represents the type of monitor to be selected for playback for a floating or full screen window.

Type: Number *Access: R/W.*

What is the difference between the **`monitor`** and **`monitorType`** properties? The **`monitor`** property specifies a specific monitor on the current running system by defining its rectangle. The **`monitorType`** specifies a general category of monitor such as primary, secondary, best color depth, and so forth. A PDF file that does not use JavaScript cannot specify a particular monitor, but it can specify a **`monitorType`**. When a **`monitorType`** is specified in a call to **`app.media.createPlayer()`** or **`app.media.openPlayer()`**, JavaScript code in `media.js` fetches the list of actual monitors available on the running system and then uses the **`monitorType`** to select one of those monitors for playback. This monitor rectangle is then used when **`MediaPlayer.open()`** is called to select the actual monitor.

Example

Play a media clip in full screen on a monitor with the best color depth from a form button.

```
var player = app.media.openPlayer({
     rendition: this.media.getRendition("Clip"),
     settings: {
            monitorType: app.media.monitorType.bestColor,
            windowType: app.media.windowType.fullScreen,
     }
});
```

page

6.0				

For a docked media player, **`MediaSettings.page`** is the document page number in which the player should be docked. For other types of media players, this property is ignored.

Type: Number *Access: R/W.*

See also **`MediaPlayer.page`**.

palindrome

6.0				

If **`MediaSettings.palindrome`** is **`true`**, the media plays once normally and then plays in reverse back to the beginning. If **`repeat`** is specified, then this forward-and-reverse playback will repeat that many times. Each complete forward and reverse playback counts as one repeat.

The default value is **`false`**.

Type: Boolean *Access: R/W.*

NOTE: Most media players do not support palindrome and ignore this setting.

Example

Use QuickTime, which supports palindrome, to view the media clip.

```
var playerList = app.media.getPlayers().select({ id: /quicktime/i });
var settings = { players: playerList, palindrome: true };
var player = app.media.openPlayer({ settings: settings });
```

The above code should be run within a Rendition action event with an associated rendition.

players

6.0				

`MediaSettings.players` is an array of objects that represent the media players that may be used to play this rendition. JavaScript code does not usually access this array directly, but passes it through from **`Rendition.select()`** to the **`settings`** object for **`app.media.createPlayer()`**.

Type: Players or Array of String *Access: R/W.*

Example

List the available players that can play this rendition. This script is run as a Rendition action with associated rendition.

```
var player = app.media.openPlayer({ settings: {autoPlay: false} });
console.println("players: " + player.settings.players.toSource() );

// Sample output to the console:
players: [{id:"vnd.adobe.swname:ADBE_MCI", rank:0},
{id:"vnd.adobe.swname:AAPL_QuickTime", rank:0},
{id:"vnd.adobe.swname:RNWK_RealPlayer", rank:0},
{id:"vnd.adobe.swname:MSFT_WindowsMediaPlayer", rank:0}]
```

rate

6.0				

`MediaSettings.rate` defines the playback rate, where 1 is normal playback, .5 is half-speed, 2 is doublespeed, -1 is normal speed in reverse, and so on.

Many players and media types are limited in the values they support for rate and will choose the closest playback rate that they support.

The default value is 1.

Type: Number *Access: R/W.*

Example

Play a media clip at doublespeed. This script is executed as a Rendition action.

```
var player = app.media.createPlayer();
player.settings.rate = 2;
player.open();
```

repeat

6.0				

`MediaSettings.repeat` is the number of times the media playback should automatically repeat. The default of value of 1 causes the media to be played once.

Many players support only integer values for repeat, but some allow non-integer values such as 1.5. A value of **`Infinity`** plays the media clip continuously.

The default value is 1.

Type: Number *Access: R/W.*

Example

Play a media clip from a Rendition action continuously.

```
var player = app.media.openPlayer({settings: { repeat: Infinity } });
```

showUI

6.0				

`MediaSettings.showUI`, a boolean, specifies whether the controls of the media player should be visible or not.

The default value is **`false`**.

Type: Boolean *Access: R/W.*

Example

Show the controls of the media player. This script is executed as a Rendition action.

```
var player = app.media.createPlayer();
player.settings.showUI = true;
player.open();
```

or

```
app.media.openPlayer( {settings: {showUI: true} });
```

startAt

6.0				

`MediaSettings.startAt` defines the starting time or frame for playback. This may be an absolute time or frame value, or a marker name, or a marker plus a time or frame, as described under MediaOffset. Playback starts at the specified time or frame, or as close to that point as the media player is able to stop. If startAt is not specified, the default value is the beginning of media.

Type: MediaOffset Object *Access: R/W.*

See also **`endAt`**.

Example

See the example that follows **`endAt.`**

visible

6.0				

`MediaSettings.visible`, a boolean, specifies whether the player should be visible.

The default value is **`true`**.

Type: Boolean *Access: R/W.*

Example

Set a docked media clip to play audio only. Script is executed as a Rendition action.

```
var args = {
     settings: {
            visible: false,
            windowType: app.media.windowType.docked
     },
};
app.media.openPlayer( args );
```

See also **`MediaPlayer.visible`**.

volume

6.0				

`MediaSettings.volume` specifies the playback volume. A value of 0 is muted, a value of 100 is normal (full) volume; values in between are intermediate volumes. Future media players may allow values greater than 100 to indicate louder than normal volume, but none currently do.

The default value is 100.

Type: Number *Access: R/W.*

windowType

6.0				

`MediaSettings.windowType` is a value chosen from the **`app.media.windowType`** enumeration, which defines what type of window the MediaPlayer should be created in.

If you use the low-level function **`doc.media.newPlayer()`**, the default value for **`windowType`** is **`app.media.windowType.docked`**.

If you use the higher-level **`createPlayer()`** or **`openPlayer()`** functions of the App.media Object, the default value is determined as follows:

If an **`annot`** is provided (see the description of the PlayerArgs Object), the default is **`app.media.windowType.docked`**.

If a settings.floating object is provided (see the description of the PlayerArgs Object), the default is **`app.media.windowType.floating`**.

Otherwise, the default is undefined.

Type: Number *Access: R/W.*

Example

The script below creates media players with different window types. Script is executed as a Rendition action, so the selection of the specification of the rendition is not needed.

```
// Docked player that will be played in the associated ScreenAnnot
app.media.openPlayer({
     settings: { windowType: app.media.windowType.docked }
});
// Play in full screen mode, see also monitor and monitorType
app.media.openPlayer({
     settings: { windowType: app.media.windowType.fullScreen }
});
// Show media clip in a floating window, also, see the floating property
var args = {
    settings: {
            windowType: app.media.windowType.floating,
            floating: {
                title: "A. C. Robat",
                width: 352,
                height: 240,
            }
        },
    };
    app.media.openPlayer( args );
```

Monitor Object

A Monitor object represents an individual display monitor. A Monitor object can be obtained from **`app.media.monitors`**, which returns an array of all monitors connected to the system. **`app.media.monitors`** is a Monitors Object so the methods of the Monitors object can be used to select or filter out monitors from a multi-monitor system based on different criteria. See the Monitors object for details.

The Monitor object and the Monitors object are used in **`MediaSettings.monitor`**.

Monitor Object Properties

colorDepth

6.0				

`Monitor.colorDepth` is the color depth of the monitor, i.e. the number of bits per pixel.

Type: Number *Access: R.*

Example

Get the primary monitor, and check its color depth. The **`Monitors.primary()`** method is use to select the primary monitor.

```
var monitors = app.monitors.primary();
console.println( "Color depth of primary monitor is "
     + monitors[0].colorDepth );
```

isPrimary

6.0				

`Monitor.primary`, a boolean, is **`true`** for the primary monitor, **`false`** for all other monitors.

Type: Boolean *Access: R.*

Example

Get the widest monitor, and see if its the primary monitor.

```
var monitors = app.monitors.widest();
var isIsNot = (monitors[0].isPrimary) ? "is" : "is not";
console.println("The widest monitor "+isIsNot+" the primary monitor.");
```

rect

6.0				

`Monitor.rect` is a rectangle representing a boundaries of the monitor in virtual desktop coordinates.

The origin of the virtual desktop origin is the top left corner of the primary monitor, so the primary monitor's bounds are always in the form [0, 0, right, bottom]. Secondary monitors may have positive or negative values in their bounds arrays, depending on where they are positioned relative to the primary monitor.

Type: Rectangle *Access: R.*

workRect

6.0				

Monitor.workRect is a rectangle representing a monitor s workspace boundaries in virtual desktop coordinates. See **`Monitor.rect`** for information about these coordinates.

The workspace is the area of a monitor that is normally used for applications, omitting any docked toolbars, taskbars, or the like. For example, running Windows on a single 800x600 display, **`Monitor.rect`** is [0, 0, 800, 600]. With a standard Windows taskbar 30 pixels high and always visible at the bottom of the screen, Monitor.workRect is [0, 0, 800, 570].

Type: Rectangle *Access: R.*

Monitors Object

A Monitors object is a read-only array of Monitor Object, each one representing a display monitor.

The **`app.monitors`** property returns a Monitors object that includes every monitor connected to the user's system. JavaScript code can loop through this array to get information about the available monitors and select one for a full screen or popup media player.

Monitors also has a number of filter methods that select one or more monitors based on various criteria.

All of the monitor selection options provided in the PDF file format are implemented as calls to these filter methods, which are written in JavaScript code in `media.js`.

None of the Monitors filter methods modify the original Monitors object. They each return a new Monitors object which normally contains one or more Monitor objects. If a single monitor matches the filtering criterion better than any other, the result Monitors object contains that one monitor. If more than one monitor satisfies the filtering criterion equally (e.g. for the **`bestColor()`** method, if more than one monitor has the same, greatest color depth), then the result contains all of those monitors.

Several of the filter methods have an optional minimum or require parameter. If this parameter is specified and no monitor meets that minimum requirement, then the result Monitors object is empty. Otherwise, the result will always contain at least one monitor, if the original Monitors object was not empty.

Wherever a filter method refers to height, width, or area, these are dimensions in pixels, not physical size.

A Monitors object is not actually an Array type, but it can be used as if it were a read-only array with numbered elements and a length property.

Monitors Object Properties

A Monitors object works like an Array, where each array element is a Monitor Object that represents a single monitor. The Monitors object returned by **`app.monitors`** is unsorted the monitors are not listed in any particular order. Elements of the Monitors object can be accessed using the usual array notation.

Example.

```
var monitors = app.monitors;
for ( var i = 0; i< monitors.length; i++)
console.println("monitors["+i+"].colorDepth = "+monitors[i].colorDepth);
```

`Monitors.length` contains the number of elements in the Monitors object. For the Monitors object returned by **`app.monitors`**, this is the number of monitors in the user's system. For a Monitors object returned by one of the filter methods, this number may be smaller.

Monitors Object Methods

bestColor

6.0				

The **`Monitors.bestColor()`** method returns a copy of the Monitors Object, filtered to include the monitor(s) with the greatest color depth.

If **`nMinColor`** is specified, returns an empty Monitors array if the best color depth is less than **`nMinColor`**.

Parameters

`nMinColor`	(optional) The minimal color depth required of the monitor.

Returns

A Monitors Object

Example

```
var monitors = app.monitors.bestColor(32);
if (monitors.length == 0 )
     console.println("Cannot find the required monitor.");
else
     console.println("Found at least one monitor.");
```

bestFit

6.0				

The **`Monitors.bestFit()`** method returns a copy of the Monitors Object, filtered to include only the smallest monitor(s) with at least the specified **`nWidth`** and **`nHeight`** in pixels.

Parameters

`nWidth`	Minimum width of the best fit monitor
`nHeight`	Minimum height of the best fit monitor.
`bRequire`	(optional) If no monitors have at least the specified width and height, then returns an empty Monitors array if **`bRequire`** is **`true`**, or a Monitors array containing the largest monitor(s) if **`bRequire`** is **`false`** or omitted.

Returns

A Monitors Object

desktop

`Monitors.desktop()` creates a new Monitors Object containing one Monitor which represents the entire virtual desktop. In this Monitor object, the **`rect`** property is the union of every **`rect`** in the original Monitors object, the **`workRect`** property is the union of every **`workRect`** in the original Monitors object, and **`colorDepth`** is the minimum **`colorDepth`** value found in the original Monitors object.

Parameters

None

Returns

A Monitors Object

NOTE: The **`desktop()`** method is normally called directly on a Monitors object returned by **`app.monitors`**. If that Monitors object is first filtered by any of its other methods, then the **`desktop()`** method does the same calculations listed above with that subset of the monitors.

document

6.0				

The **`Monitors.document()`** method returns a copy of the Monitors Object, filtered to include the monitor(s) that display the greatest amount of the document, as specified by the document object parameter **`doc`**.

If the document does not appear on any of the monitors in the original Monitors object, then returns an empty Monitors array if **`bRequire`** is **`true`** or a Monitors array containing at least one arbitrarily chosen monitor from the original array if **`bRequire`** is **`false`** or omitted.

Parameters

`doc`	The document object of the document
`bRequire`	(optional) A boolean. See the description above.

Returns

A Monitors Object

filter

6.0				

`Monitors.filter()` returns a copy of the Monitors Object, filtered by calling a ranker function for each monitor in the list. The ranker function takes a Monitor parameter and returns a numeric rank. The return value from **`filter()`** is a Monitors array containing the monitors which had the highest rank (either a single monitor, or more than one if there was a tie).

Parameters

`fnRanker`	A (ranker) function that takes a Monitor parameter and and returns a numeric rank
`nMinRank`	(optional) If **`nMinRank`** is undefined, **`filter()`** always includes at least one monitor from the original list (unless the original list was empty). If **`nMinRank`** is specified, then **`filter()`** returns an empty Monitors array if no monitors had at least that rank according to the ranker function.

Returns

A Monitors Object

NOTE: Most of the other Monitors filtering functions are implemented as filter() calls.

Example:

This is the implementation of **`Monitors.bestColor( minColor )`** from `media.js`: Returns a Monitors object containing the monitor(s) that have the greatest color depth. If minColor is specified, returns an empty Monitors array if the best color depth is less than minColor.

```
bestColor: function( minColor )
{
      return this.filter(
            function( m ) { return m.colorDepth; }, minColor );
}
```

largest

6.0				

`Monitors.largest()` returns a copy of the Monitors Object, filtered to include the monitor(s) with the greatest area in pixels.

Parameters

`nMinArea`	(optional) If the optional parameter **`nMinArea`**, a number, is specified, **`largest()`** returns an empty Monitors array if that greatest area is less than that value.

Returns

A Monitors Object

leastOverlap

6.0				

The **`Monitors.leastOverlap()`** method returns a copy of the Monitors Object, filtered to include the monitor(s) that contain the smallest amount of the rectangle, as specified by the **`rect`** parameter.

Parameters

`rect`	A rectangle, an array of four numbers in screen coordinates.
`maxOverlapArea`	(optional) If **`maxOverlapArea`** is specified, the result Monitors array contains only those monitors which contain at least that much area of the rectangle, or an empty Monitors array if no monitors contain that much area of the rectangle.

Returns

A Monitors Object

mostOverlap

6.0				

The **`Monitors.mostOverlap()`** method returns a copy of the Monitors Object, filtered to include the monitor(s) that contain the largest amount of the rectangle, as specified by the **`rect`** parameter.

Parameters

`rect`	A rectangle, an array of four numbers in screen coordinates.

`minOverlapArea`	(optional) If there is no monitor with at least that much overlapping area, then returns an empty Monitors array if `minOverlapArea` is specified, or a Monitors array containing at least one arbitrarily chosen monitor from the original array if `minOverlapArea` is omitted.

Returns

A Monitors Object

nonDocument

6.0				

The `Monitors.nonDocument()` method returns a copy of the Monitors Object, filtered to include the monitor(s) that display none of, or the least amount of the document.

Parameters

`doc`	The document object of the target document
`bRequire`	(optional) `bRequire` is a boolean which determines the return value when there is no monitor that is completely clear of the document. If `true`, `nonDocument()` returns an empty, or if false or omitted, `nonDocument()` returns a Monitors array containing at least one arbitrarily chosen monitor from the original Monitors array.

Returns

A Monitors Object

primary

6.0				

`Monitors.primary()` returns a copy of the Monitors Object, filtered by removing all secondary monitors, leaving only the primary monitor if it was present in the original list.

If the primary monitor was not present in the original list, returns a Monitors array containing at least one arbitrarily chosen monitor from the original list.

Parameters

None

Returns

A Monitors Object

Example

Get the primary monitor, and check its color depth.

```
var monitors = app.monitors.primary();
// recall that each element in a monitors object is a monitor object,
// this code uses monitor.colorDepth
console.println( "Color depth of primary monitor is "
     + monitors[0].colorDepth );
```

secondary

6.0				

Monitors.secondary() returns a copy of the Monitors Object, filtered by removing the primary monitor, returning only secondary monitors.

If the original Monitors object contained only the primary monitor and no secondary monitors, returns the original list.

Parameters

None

Returns

A Monitors Object

select

6.0				

Monitors.select() returns a copy of the Monitors Object, filtered according **nMonitor**, a monitor selection value as used in PDF and enumerated in **app.media.monitorType**.

The **doc** is required when **nMonitor** is **app.media.monitorType.document** or **app.media.monitorType.nonDocument**, and ignored for all other **nMonitor** values.

These selection values correspond directly to the various Monitors filter methods. **select()** calls the corresponding filter method, and then, in most cases, also filters with **primary()** as a tie-breaker in case more than one monitor matches the main filter. See the code in `media.js` for details.

Parameters

nMonitor	The monitor type, a number from app.media.monitorType.

doc	A document object. The parameter is required if **nMonitor** is either app.media.monitorType.document or app.media.monitorType.nonDocument, ignored otherwise.

Returns

A Monitors Object

Example:

```
// These two calls are equivalent:
settings.monitor =
app.monitors().select( app.media.monitorType.document, doc );
settings.monitor = app.monitors().document(doc).primary();
```

tallest

6.0				

Monitors.tallest() returns a copy of the Monitors Object, filtered to include only the monitor(s) with the greatest height in pixels.

Parameters

nMinHeight	(optional) If **nMinHeight** is specified and no monitor has at least that height, the return value is an empty Monitors array.

Returns

A Monitors Object

widest

6.0				

Monitors.widest() returns a copy of the Monitors Object, filtered to include only the monitor(s) with the greatest width in pixels.

Parameters

nMinWidth	(optional) If nMinWidth is specified and no monitor has at least that width, the return value is an empty Monitors array.

Returns

A Monitors Object

PlayerInfo Object

A PlayerInfo object represents a media player that is available for media playback. The **`app.media.getPlayers()`** function returns a PlayerInfoList Object, which is a collection of PlayerInfo objects.

PlayerInfo Object Properties

id

6.0				

`PlayerInfo.id` represents a media player plug-in and associated media player. This string is not localized and is not intended for display to the user. This string may be used in the **`MediaPlayer.settings.players`** array when creating a MediaPlayer, and it is also found in the **`MediaPlayer.id`** property after opening a player.

Type: String *Access: R.*

Example

List player info for all media players that play "video/mpeg".

```
var playerInfoList = app.media.getPlayers("video/mpeg");

for ( var i=0; i < playerInfoList.length; i++) {
    console.println("id: " + playerInfoList[i].id)
    console.println("name: " + playerInfoList[i].name)
    console.println("version: " + playerInfoList[i].version)
}
```

mimeTypes

6.0				

The **`PlayerInfo.mimeTypes`** property returns an array of strings listing the MIME types that this media player supports.

Type: Array of String *Access: R.*

Example:

```
var qtinfo = app.media.getPlayers().select({id: /quicktime/i })[0];
console.println( qtinfo.mimeTypes );
```

name

6.0				

`PlayerInfo.name` is the name of the media player. This string is localized according to the current language as found in app.language. It is suitable for display in list boxes and the like, but not for direct comparisons in JavaScript code.

Type: String *Access: R.*

version

6.0				

`PlayerInfo.version` is a string containing the version number of the media player. For most players, it is the version number of the underlying media player that is installed on the user's system. This string is in dotted decimal format, e.g. 7.4.030.1170 .

Type: String *Access: R.*

PlayerInfo Object Methods

canPlay

6.0				

`PlayerInfo.canPlay()` checks to see if the media player can be used for playback, taking the user's security settings into account.

If the parameter **`bRejectPlayerPrompt`** is **`true`**, then the method returns **`false`** if using this player would result in a security prompt. Otherwise the method returns **`true`** if playback is allowed either with or without a security prompt. (This method itself never triggers a security prompt, but a later attempt to create a media player may.)

Parameters

`oDoc`	A document object
`bRejectPlayerPrompt`	A boolean, which if **`true`**, the method returns **`false`** if using this player would result in a security prompt, and if **`false`**, returns **`true`** if playback is allowed either with or without a security prompt. The default is **`false`**.

Returns

Boolean

canUseData

6.0				

Tells whether the player can use the specified data, as passed by its parameter **`oData`**, for playback. Returns **`true`** if the data can be used for playback, and **`false`**, otherwise.

Parameters

`oData`	**`oData`** is a MediaData object (see **`MediaSettings.data`** for a description of this object). This data object is obtained in several ways, from **`app.media.getAltTextData()`**, **`app.media.getURLData()`**, or indirectly via **`Rendition.getPlaySettings()`**.

Returns

Boolean

PlayerInfoList Object

The **`app.media.getPlayers()`** method returns a PlayerInfoList, which is an array of PlayerInfo Objects.

The PlayerInfoList has one method, **`select()`**, which can be used to filter the list using any of the properties in a PlayerInfo.

When a media player is created using **`app.media.createPlayer()`**, the **`settings.players`** property (see the PlayerArgs Object) may contain a PlayerInfoList, to restrict the player creation to choose from only those players specified in the list.

PlayerInfoList Object Properties

The PlayerInfoList works like an array, the elements, which are PlayerInfo Objects, can be accessed using the usual array notation. The number of elements in the PlayerInfoList array can be obtain from the **`length`** property.

PlayerInfoList Object Methods

select

6.0				

`PlayerInfoList.select()` returns a copy of the PlayerInfoList, filtered to include only the players that meet the selection criteria. This will be an empty array if no players match.

The object parameter may contain any of the following properties. Any properties that are specified are required to match; properties that are omitted match any player.

```
id: string or regular expression
name: string or regular expression
version: string or regular expression
```

The id, name, and version properties may be either strings for an exact match, or regular expressions.

Parameters

`object`	(optional) An object which contains any of the properties **`id`**, **`name`**, or **`version`**. The values of these properties may be a string or a regular expression.

Returns

PlayerInfoList Object

Example 1

Use QuickTime to view the media clip.

```
var playerList = app.media.getPlayers().select({ id: /quicktime/i });
// QuickTime supports palindrome, so let's try it.
var settings = { players: playerList, palindrome: true };
var player = app.media.openPlayer({ settings: settings });
```

Example 2

Choose the Flash player by using a pattern match on its player ID.

```
var player = app.media.createPlayer();
player.settings.players = app.media.getPlayers().select({ id:/flash/i});
player.open();
```

Rendition Object

A Rendition contains information needed to play a media clip, including embedded media data (or a URL) and playback settings. It corresponds to a Rendition in the Acrobat authoring user interface.

A Rendition is a base type for either a MediaRendition or a MediaSelector. A function that accepts a Rendition can take either of these two types. The properties and methods described in this section are available for both MediaRendition and MediaSelector. Use Rendition.type to distinguish between MediaRendition and MediaSelector.

Rendition Object Properties

altText

6.0				

The **`rendition.altText`** property is the alternate text string for the rendition (an empty string if no alternate text was specified). This property is available only if the **`type`** of the **`rendition`** is **`app.media.renditionType.media`** (a MediaRendition).

Type: String *Access: R.*

Example

Get the **`altText`** of a rendition.

```
this.media.getRendition("myClip").altText;
```

See the examples that follow **`app.media.getAltTextSettings()`**

doc

6.0				

The **`Rendition.doc`** property is a reference to the document that contains the Rendition.

Type: Doc *Access: R.*

fileName

6.0				

The **`rendition.fileName`** property returns an empty string if the media is embedded, and the filename or URL of the media if it's not embedded. This property is available only if the **`type`** of the **`rendition`** is **`app.media.renditionType.media`**.

Type: String *Access: R.*

type

6.0				

The **`Rendition.type`** is an **`app.media.renditionType`** value indicating the type of rendition.

Currently, there are two types: MediaRendition and RenditionList:

When **`Rendition.type`** is equal to **`app.media.renditionType.media`**, the Rendition is a MediaRendition. A MediaRendition is an individual Rendition, as it appears in the Settings tab of the Multimedia Properties dialog of the UI.

When **`Rendition.type`** is equal to **`app.media.renditionType.selector`**, the Rendition is a RenditionList. A RenditionList is an array of MediaRendition. The list is the one that appears in the Settings tab of he Multimedia Properties dialog of the UI.

Future versions of Acrobat may add more **`renditionType`** values, so JavaScript code should not assume that only the existing **`app.media.renditionType`** values may be encountered.

Type: Number *Access: R.*

uiName

6.0				

`Rendition.uiName` contains the name of the Rendition as found in the **N** entry in its dictionary in the PDF file.

Type: String *Access: R.*

Example

The following is executed as a Rendition action.

```
console.println("Preparing to play \""
     + event.action.rendition.uiName + "\"");
```

See the Event Object for a description of **`event.action.rendition`**.

Rendition Object Methods

getPlaySettings

6.0				

Creates and returns a MediaSettings Object that can be used to create a MediaPlayer object.

This method is available only for a MediaRendition.

Parameters

`bGetData`	(optional) A boolean, which if **`true`**, the MediaSettings object returns the MediaData (See MediaSettings.data).

Returns

A MediaSettings Object

NOTE: **`app.media.getAltTextSettings()`** calls **`getPlaySettings(false)`** to obtain the correct settings to display alternate text, see the `media.js`.

This MediaSettings object includes these properties:

```
autoPlay
baseURL (if specified in rendition)
bgColor
bgOpacity
data (if bGetData is true)
duration
endAt
layout
monitorType
palindrome
showUI
rate
repeat
startAt
visible
volume
windowType
```

In the current version of Acrobat, all of these properties are present in the settings object (except as noted above), and **`null`** is used when values such as **`startAt`** are unspecified. This may change in the future to return only those values which are actually specified, with defaults assumed for the rest.

Example:

```
// Get the MediaSettings for this Rendition
```

```
var settings = myRendition.getPlaySettings();
if( settings.startAt !== null ) // Do NOT use this strict comparison!
...
if( settings.startAt ) // This is OK
...
```

See **`app.media.getAltTextSettings()`** and **`app.media.openPlayer()`** for examples of usage.

select

6.0				

`Rendition.select()` selects a media player to play a MediaRendition or a RenditionSelector. If the Rendition is a RenditionSelector, **`select()`** examines every MediaRenditon contained within and selects the most suitable one. (See **`type`** for a description of RenditionSelector and MediaRendition.)

The return value is a MediaSelection Object that can be used to create a MediaSettings Object. This object can then be used to create a MediaPlayer Object.

Parameters

`bWantRejects`	(optional) If **`bWantRejects`** is **`true`**, the **`rejects`** property of the resulting MediaSelection will contain information about media players that were rejected during the selection process.
`oContext`	(optional) **`oContext`** is a MediaSelection.selectContext value from a previous **`Rendition.select()`** call. This parameter allows you to write a loop that calls **`Rendition.select()`** repeatedly until you find a media player that satisfies any selection criteria that you want to test in JavaScript code.

Returns

A MediaSelection Object

Example 1

Get a usable MediaSelection for this Rendition

```
var selection = rendition.select();
```

Example 2

Get the name of the selected rendition. This script is executed from a Rendition action event.

```
var selection = event.action.rendition.select();
console.println( "Preparing to play " + selection.rendition.uiName);
```

testCriteria

6.0				

This method tests the Rendition against any criteria that are specified in the PDF file, such as minimum bandwidth, and returns a boolean indicating whether the Rendition satisfied all of those criteria.

Parameters

None

Returns

Boolean

ScreenAnnot Object

A ScreenAnnot is a rectangular area within a PDF document viewed on the display screen. A ScreenAnnot may have Renditions and RenditionActions associated with it for multimedia playback.

ScreenAnnot Object Properties

altText

6.0				

The **`annot.altText`** property is the alternate text string for annot (an empty string if no alternate text was specified).

Type: String *Access: R.*

Example

Get an annot and write its **`altText`** to the debug console.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
console.println( "annot.altText = " + annot.altText );
```

alwaysShowFocus

6.0				

Normally, a ScreenAnnot shows and hides a focus rectangle to indicate whether it has the keyboard focus. If **`ScreenAnnot.alwaysShowFocus`** is **`true`**, the focus rectangle is displayed by the ScreenAnnot even if it does not have the focus. This is used for docked media playback, so that the focus rectangle of the annot can remain visible even though the media player actually has the keyboard focus.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Boolean *Access: R/W.*

display

6.0				

Same as **`Field.display`**, as documented in the *Acrobat JavaScript Scripting Reference*.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Integer *Access: R/W.*

Example

Hide the annot

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
annot.display = display.hidden;
```

doc

6.0				

`ScreenAnnot.doc` is a reference to the document that contains the ScreenAnnot.

Type: Doc object *Access: R.*

events

6.0				

`ScreenAnnot.events` is an Events Object containing the event listeners that are attached to a ScreenAnnot.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Events Object *Access: R/W.*

Example

Create a simple focus event listener.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
var myFocusEvent = {
     onFocus: function () {
            console.println("Focusing...");
     }
};
annot.events.add( myFocusEvent );
```

This event listener can be removed at a later time by executing the following code.

```
annot.events.remove( myFocusEvent );
```

extFocusRect

6.0				

When a ScreenAnnot draws a focus rectangle, the rectangle normally encloses only the ScreenAnnot itself. If **`extFocusRect`** is specified, then the ScreenAnnot takes the union of its normal rectangle and **`extFocusRect`**, and it uses the resulting rectangle to draw the focus rectangle.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Array of Number of length 4 *Access: R/W.*

innerDeviceRect

6.0				

`ScreenAnnot.innerDeviceRect` and **`ScreenAnnot.outerDeviceRect`** define the interior and exterior rectangles of the ScreenAnnot as it appears in the current page view.

Type: Array of Number of length 4 *Access: R.*

Example

Get the **`innerDeviceRect`**.

```
annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
console.println("annot.innerDeviceRect = "
     + annot.innerDeviceRect.toSource() );
```

noTrigger

6.0				

If **ScreenAnnot.noTrigger** is **true**, then the screen annot cannot be triggered through the Acrobat user interface. Typically, clicking the mouse on a Screen Annot starts playback of a media player; noTrigger suppresses this.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Boolean *Access: R/W.*

Example

Use form buttons to control the media clip, so turn off interaction with annot.

```
annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
annot.noTrigger = true;
```

outerDeviceRect

6.0				

ScreenAnnot.innerDeviceRect and **ScreenAnnot.outerDeviceRect** define the interior and exterior rectangles of the ScreenAnnot as it appears in the current page view.

Type: Array of Number of length 4 *Access: R.*

page

6.0				

ScreenAnnot.page is the page number of the PDF file in which the ScreenAnnot is located.

Type: Number *Access: R.*

player

6.0				

ScreenAnnot.player is a reference to the MediaPlayer associated with a ScreenAnnot. This property exists only for a ScreenAnnot Object that is connected to a MediaPlayer. The property is set by **MediaPlayer.open()** or by methods that call **open()** indirectly, such as **app.media.openPlayer()**.

Type: ScreenAnnot *Access: R/W.*

rect

6.0	Ⓓ			

`ScreenAnnot.rect` is the rectangle of the ScreenAnnot in default user coordinates. Changing this property dirties the PDF file, and the new setting will be saved if the PDF file is saved. The **`innerDeviceRect`** and **`outerDeviceRect`** properties are also updated to reflect the new rectangle.

Type: Array of Number of length 4 *Access: R/W.*

Example

Adjust the position of the annot slightly.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
var aRect = annot.rect;
aRect[0] += 10;
aRect[2] += 10;
annot.rect = aRect;
```

ScreenAnnot Object Methods

hasFocus

6.0				

`ScreenAnnot.hasFocus()` tells whether the screen annot currently has the keyboard focus.

Parameters

None

Returns

Boolean

setFocus

6.0				

`ScreenAnnot.setFocus()` sets the keyboard focus to the screen annot. The focus is set synchronously (before setFocus returns) if it is safe to do so. If it is unsafe to set the focus synchronously (e.g. when the property is changed within an on event method), then

bAllowAsync determines what happens: If **true**, the focus will be set asynchronously during idle time; if **false** or omitted, the focus remains unchanged.

The return value is **true** if the operation was performed synchronously, or **false** if it was deferred to be performed asynchronously.

Parameters

bAllowAsync	(optional) A boolean which determines the behavior of **setFocus()** when it is not safe to set the focus synchronously. If **true**, the focus will be set asynchronously during idle time; if **false** or omitted, the focus remains unchanged. The default is **false**.

Returns

Boolean