Windows Media Player 9 Series SDK

This documentation describes the Microsoft® Windows Media® Player 9 Series Software Development Kit (SDK). The Windows Media Player SDK is one of the main components of the Microsoft Windows Media SDK. Other components include the Microsoft Windows Media Format SDK, the Microsoft Windows Media Services SDK, the Microsoft Windows Media Encoder SDK, and the Microsoft Windows Media Rights Manager SDK.

The Windows Media Player SDK documents programming technologies that can be used to extend the capabilities of Windows Media Player. These technologies are documented in the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Windows Media Player</td>
<td>This section provides details about how to find specific information in the</td>
</tr>
<tr>
<td>SDK</td>
<td>SDK. It includes a section about new features and information about how to</td>
</tr>
<tr>
<td></td>
<td>use the samples included with the SDK.</td>
</tr>
<tr>
<td>Windows Media Player Object</td>
<td>The Microsoft Windows Media Player control is a Microsoft ActiveX® control</td>
</tr>
<tr>
<td>Model</td>
<td>used for adding multimedia playback capabilities to Web pages. It provides</td>
</tr>
<tr>
<td></td>
<td>a programming interface for rendering a variety of network streaming and</td>
</tr>
<tr>
<td></td>
<td>non-streaming multimedia formats.</td>
</tr>
<tr>
<td>Windows Media Player Skins</td>
<td>Skins are an XML-based technology used to customize the user interface of</td>
</tr>
<tr>
<td></td>
<td>the stand-alone Windows Media Player.</td>
</tr>
<tr>
<td>Windows Media Player Plug-ins</td>
<td>Plug-ins are objects that extend Windows Media Player functionality in a</td>
</tr>
<tr>
<td></td>
<td>variety of ways. Plug-in types include custom visualizations, user</td>
</tr>
<tr>
<td></td>
<td>interface plug-ins, DSP plug-ins, and rendering plug-ins.</td>
</tr>
<tr>
<td>Windows Media Metafiles</td>
<td>Metafiles are XML documents that provide information about a media stream</td>
</tr>
<tr>
<td></td>
<td>and its presentation. Metafiles can be used to organize media files into</td>
</tr>
<tr>
<td></td>
<td>playlists that can include functionality for seamless stream switching, ad</td>
</tr>
<tr>
<td></td>
<td>insertion, and other features.</td>
</tr>
<tr>
<td>Windows Media Player</td>
<td>Some features of the SDK apply to the Windows Media Player application and</td>
</tr>
<tr>
<td></td>
<td>the ActiveX control in general. This section provides information about</td>
</tr>
<tr>
<td></td>
<td>these general features.</td>
</tr>
<tr>
<td>Glossary</td>
<td>This section contains definitions of terms used throughout the SDK.</td>
</tr>
</tbody>
</table>

Notes Installing this SDK does not install Windows Media Player. You must have Windows Media Player installed to use the material in this SDK.
About the Windows Media Player SDK

The Microsoft Windows Media Player 9 Series Software Development Kit (SDK) provides information and tools to customize Windows Media Player and to use the Windows Media Player ActiveX control. This documentation also provides information about using Windows Media metafiles.

Support for customizing Windows Media Player is provided by:

- **Windows Media Player skins.** Skins allow you both to customize the Player user interface and to enhance its functionality by using XML.
- **Windows Media Player plug-ins.** Windows Media Player includes support for plug-ins that create visualization effects, that perform digital signal processing (DSP) tasks, that add custom user interface elements to the full mode Player, and that render custom data streams in digital media files created using the ASF file format.

Embedding the Windows Media Player control is supported for a variety of technologies, including:

- HTML in Web browsers. Microsoft Internet Explorer and Netscape Navigator version 4.7, 6.2, and 7.0 browsers are supported.
- Programs created with the Microsoft Visual C++® development system.
- Programs based on Microsoft Foundation Classes (MFC).
- Programs created with Microsoft Visual Basic® 6.0.
- Programs created using the .NET Framework, including programs written in the C# programming language.
- Microsoft Office.

This overview contains the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Organization of the Documentation</strong></td>
<td>Explains where to find information about specific technologies in the SDK documentation.</td>
</tr>
<tr>
<td><strong>What's New</strong></td>
<td>Details the features that have been added to the SDK since the previous version.</td>
</tr>
<tr>
<td><strong>Samples</strong></td>
<td>Describes the code samples installed with the SDK and where to find them.</td>
</tr>
<tr>
<td><strong>For More Information</strong></td>
<td>Describes where to find the latest white papers and articles on the Microsoft Web site.</td>
</tr>
</tbody>
</table>

See Also

- **Windows Media Player 9 Series SDK**
The Windows Media Player SDK documents a wide variety of technologies related to Windows Media Player. To make it easier to locate the information you need, the documentation is divided into sections that help you quickly locate topics about a particular feature or technology.

Where appropriate, major sections each include three subsections: About, Programming Guide, and Reference. The About section describes the particular feature or technology to help you understand how you can use it. The Programming Guide section adds tutorials that take you deeper into the technology with step-by-step instructions. The Reference section details the interfaces, objects, methods, properties, events, elements, and attributes that are available for your use.

The following table will help you to understand where to look for the help you need.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Windows Media Player SDK</td>
<td>Provides overviews of the SDK's organization and of the new features and technologies available. Provides information about where to find the SDK samples and where to locate more help on the Web.</td>
</tr>
<tr>
<td>Windows Media Player Object Model</td>
<td>Provides detailed information about using the Windows Media Player ActiveX control object model. The About the Player Object Model section describes each object and provides general information about the control. The Player Control Guide includes topics about how to write code for the embedded Player, as well as an Object Model Migration Guide that details the differences between this version of the control and the Windows Media Player 6.4 ActiveX control. The Object Model Reference section is written in a style designed to be easily understood by script programmers and Visual Basic 6.0 programmers. A C++ Translation Guide is included to help C++ programmers use the reference documentation as well. C++-only interfaces are documented in the Object Model Reference for C++.</td>
</tr>
<tr>
<td>Windows Media Player Skins</td>
<td>Provides detailed information about creating Windows Media Player skins. The About Skins</td>
</tr>
</tbody>
</table>
The **Skin Creation Guide** section teaches you to make your first skin and shows you how to add particular features. The **Skin Programming Reference** details the elements and attributes available to skins. Skins can also use the Microsoft JScript® programming language to add functionality, so the Object Model Reference section is useful to skin programmers as well.

Windows Media Player Plug-ins

There are four basic types of Windows Media Player plug-ins. Each type is detailed in a subsection of this section.

Windows Media Player Custom Visualizations

Visualizations are a plug-in technology that you can use to create interesting visual effects for the Player. About Custom Visualizations describes the technology and tells you what you need to know to create a visualization using the plug-in wizard. The Custom Visualization Programming Guide steps you through the process of changing the code generated by the plug-in wizard to create a different visualization. The Custom Visualization Programming Reference section provides detailed information about the interfaces, structures, and enumerations you need to write visualization code.

Windows Media Player User Interface Plug-ins

User interface (UI) plug-ins allow you to add new user interface elements to the full mode Player. The About User Interface Plug-ins section provides an overview of UI plug-ins and describes how to use the plug-in wizard to create a plug-in. The User Interface Plug-ins Programming Guide teaches you to modify the code generated by the plug-in wizard to create a new UI plug-in. The User Interface Plug-ins Programming Reference section provides detailed information about the interfaces you need to write UI plug-in code.

Windows Media Player DSP Plug-ins

Digital Signal Processing (DSP) plug-ins allow you to write code that changes the audio or video data. The About DSP Plug-ins section provides the information you need to understand the technology and use the plug-in wizard to create a sample plug-in. The DSP Plug-ins Programming Guide teaches you to modify the code generated by the plug-in wizard to create a new audio DSP effect plug-in. The DSP Plug-ins Programming Reference section provides detailed information about the interfaces and enumerations you need to write DSP plug-in code.

Windows Media Player Rendering Plug-ins

Rendering plug-ins allow you to write code that renders custom binary data delivered from a
What's New

The Windows Media Player 9 Series SDK introduces a range of new features and functionality for customizing Windows Media Player and using the Player control. Some of the features represent entirely new Player technologies; others extend and update existing technologies to enable new scenarios. The following sections provide an overview changes made to the SDK since the previous version.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's New for the Windows Media Player Control</td>
<td>Describes new features related to the Windows Media Player ActiveX control.</td>
</tr>
</tbody>
</table>
What's New for the Windows Media Player Control

The Windows Media Player ActiveX control adds new functionality in Windows Media Player 9 Series. It also updates and extends existing functionality from the previous version. New and updated features include:

- Enhanced and updated Media Library support. See Working with Media Library.
- Branding and advertising support for Internet content providers (ICPs). See Displaying Web Pages in Windows Media Player and Logging Stream Data.
- Language selection. There are new properties and methods that enable you to take advantage of multiple language tracks. See Controls Object.
- DVD support. To complement the existing functionality available in the Cdrom object, the Player object model adds the new DVD object. The DVD object includes properties and methods that provide DVD-specific functions, like DVD.titleMenu and DVD.domain. See DVD Object.
- Time compression. There is new behavior for the Windows Media Player control when playing in fast forward or fast reverse. Content now scans at the specified speed with audio playback, allowing the user to review material such as newscasts or informational videos more quickly, or to slow down playback to understand details more clearly.
- Invisible user interface (UI) mode. There is a new value for the Player.uiMode property called "invisible". See Player.uiMode.
- Support for SMPTE time code. Time code is the industry standard way of identifying individual video frames. See Controls.currentPositionTimecode.
- Enhancements to closed captioning support. The ClosedCaption object exposes new methods and properties for working with languages and styles in Synchronized Accessible Media Interchange (SAMI).
files. See ClosedCaption Object.
- Support for Remoting. Remoting is a new feature available to developers working in C++ that enables the creation of applications that coordinate playback between an embedded Windows Media Player control and the full mode of the Player. When remoting, the embedded control and the full mode Player share the same playback engine. To the user, this means that video viewed in an embedded Player can seamlessly transition to the full mode Player. See PlayerApplication Object and Remoting the Windows Media Player Control.
- Support for embedding in containers other than Web browsers. See Player Control Guide.
- Using skins with the ActiveX control. See Using Skins with the Windows Media Player Control.

See Also

- What's New

What's New for Windows Media Player Skins

Windows Media Player skins offer additional functionality in Windows Media Player 9 Series. This new functionality comes in the form of new elements, attributes, and methods. These include:

- New ambient attributes for accessibility, like accKeyboardShortcut. See Ambient Attributes.
- A new ambient method and a new ambient attribute for alpha blending. See AmbientAttributes.alphaBlendTo and AmbientAttributes.alphaBlend.
- New ambient event attributes for screen height and width. See event.screenHeight and event.screenWidth.
- New elements for displaying Media Library information, such as ITEM. See Skin Programming Reference.
- New elements for displaying data, like EDITBOX, and POPUP. See Skin Programming Reference.
- A new element that displays the Quick Access Panel menu from the full mode of the Player. See AUTOMENU Element.
- New predefined elements based upon BUTTON, EFFECTS, PLAYLIST, SLIDER, TEXT, and VIDEO. See Skin Programming Reference.
- New attributes and methods for buttons, button groups, effects, equalizer settings, playlists, and views. For instance, BUTTONVIEW, PLAYLIST, VIEW, and SUBVIEW each contain attributes that enable you to change skin colors dynamically. EQUALIZERSETTINGS includes new attributes that allow you to take advantage of advanced audio processing functionality, like enabling normalization and cross-fading. See Skin Programming Reference.

See Also

- What's New
What's New for Windows Media Metafiles

The Windows Media Player 9 Series SDK includes three main changes for Windows Media metafiles. These are:

- Predefined PARAM names such as HTMLView, which lets you specify Web pages to display in the full mode of the Player. See PARAM Element.
- Additional advertising support. Still images and Shockwave Flash files are now supported by Windows Media Player, and can be used in playlist entries to provide interstitial advertisements. You can also use the log predefined PARAM name to add custom field/value pairs to the logging information already sent by the Player. See Logging Stream Data.
- The LOGO element is no longer supported.

See Also

- What's New

What's New for Windows Media Player Plug-ins

Windows Media Player 9 Series adds a new plug-in architecture. Plug-ins are add-ins for the Player that provide added or customized functionality for the end user. Types of Windows Media Player plug-ins include:

- User interface (UI) plug-ins. This architecture allows you to add new user interface elements to the full mode Player using Microsoft Windows®-based programming techniques. See Windows Media Player User Interface Plug-ins.
- Digital signal processing (DSP) plug-ins. Based on Microsoft DirectX® Media Object (DMO) architecture, DSP plug-ins allow you to insert audio or video processing code into the Player signal chain.
before rendering. See Windows Media Player DSP Plug-ins.
- Rendering plug-ins. Also based on DMO architecture, rendering plug-ins allow you to render data from a Windows Media arbitrary data stream. This allows you to use Windows Media Player to play back content in any format you can create. See Windows Media Player Rendering Plug-ins.

Although they have existed in previous versions, Windows Media Player visualizations are now considered a type of plug-in. See Windows Media Player Custom Visualizations.

See Also

- What's New
The Windows Media Player 9 Series SDK includes a variety of samples that demonstrate many of the programming techniques described in this document.

Scripting samples demonstrate how to embed the Player ActiveX control in a Web page. A complete Web page sample demonstrates how to detect the current version of the Player ActiveX control.

A complete skins sample demonstrates how to change the visual appearance and the behavior of Windows Media Player. A sample Windows Media Player border is provided.

A Visual Basic 6.0 sample project demonstrates how to create a program that can access and play back video from a DVD.

A wizard for Microsoft Visual C++ can create a working sample of each type of plug-in: a visualization plug-in, a user interface plug-in, a digital signal processing plug-in, and a rendering plug-in.

Samples for Microsoft Visual C++ demonstrate embedding the Player ActiveX control, remoting the Player ActiveX control, and using Media Library.

A C# sample demonstrates using the Player ActiveX control with the .NET Framework.

You must first install the Windows Media Player SDK to use the samples. If you have not already installed it, click the following link:

Install Windows Media Player SDK

After installing the SDK, to see descriptions of the samples and links to the sample code, click the Start button, then point to Programs, then Windows Media, then Windows Media SDK, and then click Player SDK samples.

See Also

- About the Windows Media Player SDK

For More Information

- For more information about Windows Media, see the Windows Media Technologies Web site.
- For more information about programming with Microsoft technologies, including Windows Media technologies and software development kits, see the MSDN Library Web site.
- For more information about Windows Media Player, see the Windows Media Player page on the Windows Media Technologies Web site.
# Windows Media Player Object Model

The Microsoft Windows Media Player ActiveX control object model provides scripting interfaces to give Web developers the opportunity to add Windows Media Player functionality to Web pages. By embedding the control in an HTML page, the developer can use Internet Explorer to craft visually complex graphical environments that take advantage of a rich and dynamic event model. The developer can completely sculpt the Web site user's audio and video experience.

The Windows Media Player ActiveX control object model documentation is presented in the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Player Object Model</td>
<td>Discusses the architecture of the object model in abstract terms.</td>
</tr>
<tr>
<td>Player Control Guide</td>
<td>Explains how to use the control object model.</td>
</tr>
<tr>
<td>Object Model Reference</td>
<td>Describes the methods, properties, and events that the object model makes available to scripting languages, Visual Basic 6.0, and .NET Framework languages.</td>
</tr>
<tr>
<td>Object Model Reference for C++</td>
<td>Explains how to translate the control object model reference into C++ terms. Also documents the interfaces and enumeration types that are available only through C++. Read this section if you are programming with C++.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player 9 Series SDK
About the Player Object Model

The Microsoft Windows Media Player control is a standard ActiveX control that uses Microsoft Component Object Model (COM) technology. The Windows Media Player functionality is distilled into a set of programming interfaces that follows standard COM guidelines. You can program these interfaces on a standard HTML Web page using the player control object model with Microsoft JScript or Microsoft Visual Basic Scripting Edition (VBScript). You can also program them in skins using Microsoft JScript. If you want to create a custom program that embeds the control, you can use one of several languages, including Visual Basic, C++, and C#.

Except where noted, example code in this documentation uses JScript syntax.

The following sections provide overview information about the Windows Media Player object model.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player Object Model</td>
<td>This section describes the objects available in the object model.</td>
</tr>
<tr>
<td>About the Object Model Versions</td>
<td>This section details which object model items have been added in each version since the introduction of Windows Media Player 7.0.</td>
</tr>
<tr>
<td>Properties, Methods, and Events</td>
<td>This section explains properties, methods, and events.</td>
</tr>
<tr>
<td>Supported Protocols and File Types</td>
<td>This section lists the protocols and file types supported by Windows Media Player and the Player control object model.</td>
</tr>
<tr>
<td>About Media Library</td>
<td>This section describes the information in Media Library.</td>
</tr>
<tr>
<td>Using HTML with Windows Media Player</td>
<td>This section describes the various ways to make Windows Media Player and HTML work together.</td>
</tr>
<tr>
<td>Embedding the Player Control</td>
<td>This section describes the various ways to embed the Player control in Microsoft Office documents and in custom programs.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player Object Model
Player Object Model

ActiveX uses the concept of objects to contain programming functionality. Windows Media Player uses several objects to divide up the work that the control does. The root object is the Player object, and the other objects are attached to the Player object through specific properties.

The following diagram shows how the Windows Media Player 9 Series ActiveX control object model works:

The following sections provide conceptual overviews for each object:

- About the Cdrom and CdromCollection Objects
- About the ClosedCaption Object
- About the Controls Object
- About the DVD Object
- About the Error and ErrorItem Objects
- About the External Object
- About the MediaCollection and Media Objects
- About the MetadataPicture Object
- About the MetadataText Object
- About the Network Object
- About the Player Object
- About the PlayerApplication Object
About the Cdrom and CdromCollection Objects

The **Cdrom** and **CdromCollection** objects govern the interface to the CD-ROM drives in your computer for purposes of reading and playing compact discs.

The **CdromCollection** object is accessed only through the **cdromCollection** property of the **Player** object. The **cdromCollection** property returns the **CdromCollection** object. You can only access the properties of the **CdromCollection** object after you have created it. For example, to use the **count** property, you must use the following code:

```javascript
mycount = player.cdromcollection.count;
```

You can only access the **Cdrom** object through the **CdromCollection** object. For example, to eject the CD by using the **Eject** method, you must first create the collection object and then an item in the object. To eject the CD, use the following code:

```javascript
player.cdromcollection.item(0).eject();
```

In both cases, you are first creating the collection object (**CdromCollection**) and then getting a specific object of that collection. The object is the first item in the collection, **Item(0)**, which corresponds to the first CD drive. You then call a method, **Eject**, on that item.

See Also

- **Cdrom Object**
- **CdromCollection Object**
- **Player Object Model**
About the ClosedCaption Object

The ClosedCaption object governs the captioning interface for Windows Media Player. The ClosedCaption object is obtained through the closedCaption property of the Player object. For example, to get the SAMIFileName property value, type the following:

```
myfile = player.closedcaption.SAMIFileName;
```

See Also

- ClosedCaption Object
- Player Object Model

About the Controls Object

The Controls object governs the transport of the media content through the control by using methods such as Play and Stop. It is accessed only through the Controls property of the Player object. The Controls property returns the Controls object. You can only access the properties of the Controls object after you have created it. For example, to use the Play method, you must use the following code:

```
player.controls.play();
```

See Also

- Controls Object
- Player Object Model
About the DVD Object

The DVD object adds functionality specific to DVD media. In a general sense, DVD media is treated just like other digital media in Windows Media Player. For instance, DVD-ROM drives are enumerated using the CdromCollection object and DVD titles and chapters are manipulated using Playlist objects and Media objects. Some functionality is DVD-specific, however, and the DVD object provides this. For example, DVD has a concept called domain. To retrieve the current domain for DVD media, type the following:

```plaintext
mydomain = player.dvd.domain;
```

See Also

- DVD Object
- Player Object Model

About the Error and ErrorItem Objects

The Error and ErrorItem objects govern the error-handling capabilities of Windows Media Player. The Error object is obtained from the Player object through the error property. You can get a specific code from the Error object by using the item property of the Error object to create the ErrorItem object. For example, to get the error code of the first error item, type:

```plaintext
myerrorcode = player.error.item(0).errorCode;
```

You can also invoke Web Help with the Error object by using the following code:

```plaintext
player.error.webHelp();
```

See Also

- Error Object
- ErrorItem Object
About the External Object

The **External** object can provide functionality to Web pages displayed in Windows Media Player.

See Also

- **External Object**
- **Player Object Model**

About the MediaCollection and Media Objects

The **MediaCollection** and **Media** objects govern the media collection, which defines the locations of media files that Windows Media Player can access. You get the **MediaCollection** object from the **mediaCollection** property of the **Player** object. The **mediaCollection** property returns the **MediaCollection** object. You can only access the properties of the **MediaCollection** object after you have created it. For example, to add a **Media** object (a song), type the following:

```
player.mediacollection.add('laure.wma');
```

You have added the file laure.wma to the media collection.

You can get the current **Media** object by using the **currentMedia** property of the **Player**. For example, to get the **duration** property of the current **Media** object, type:

```
myduration = player.currentmedia.duration;
```
There are many ways to get a **Media** object so that you can access its properties. For example, if you want to access the **duration** property of the current media, each of the following lines could be used:

To get the duration of the currently playing media, type:

```javascript
player.currentMedia.duration;
```

To get the duration of the current media in a playlist, type:

```javascript
player.controls.currentItem.duration;
```

To get the duration of the third media item in a playlist, type:

```javascript
player.currentPlaylist.item(2).duration;
```

To get the duration of the third media item in a "Jazz" genre, type:

```javascript
player.mediaCollection.getByGenre("jazz").item(2).duration;
```

To get the duration of the third media item in the second playlist, type:

```javascript
player.playlistCollection.getAll.item(1).item(2).duration;
```

See Also

- **Media Object**
- **MediaCollection Object**
- **Player Object Model**

About the MetadataPicture Object

The **WM/Picture** metadata attribute provides detailed information about metadata images, such as album art. The **MetadataPicture** object provides a way to retrieve the individual data related to the image. You might think of the **WM/Picture** metadata attribute as a compound attribute. When you use **Media.getItemInfoByType** to retrieve information about a metadata image, you retrieve a **MetadataPicture** object that has properties which contain the individual data.

See Also

- **MetadataPicture Object**
About the MetadataText Object

Some metadata attributes, such as WM/Lyrics_Synchronised, provide text accompanied by a description of the text. The MetadataText object provides a way to retrieve the text and the description separately. When you use Media.getItemInfoByType to retrieve information about complex textual attributes, you retrieve a MetadataText object that has properties, which contain the individual data.

See Also

- MetadataText Object
- Player Object Model

About the Network Object

The Network object governs the properties that allow you to determine how well the content is streaming through the network. For example, you can find out whether packets are being lost and take appropriate action. The Network object is accessed only through the network property of the Player object. The network property returns the Network object. You can only access the properties of the Network object after you have created it. For example, to use the Bandwidth property, you must use the following code:

```javascript
mybandwidth = player.network.bandwidth;
```

See Also

- Network Object
About the Player Object

The **Player** object is the core object for the Windows Media Player control. All other related objects are connected to this object through specific properties that return the object. For example, the **Settings** object is accessed through the **settings** property. The **Player** object provides methods, properties, and events that relate to the core functionality of Windows Media Player.

Because this reference is also to be used for skins programming, the ID of the **Player** object will be "player" for syntactical examples.

Using the **player** global attribute within a skin definition file gives access to the **Player** object for use in scripting. Through the **Player** object, all other objects in the Windows Media Player control become accessible to scripts as well. The events of the **Player** object and the **URL** property can also be specified at design time using the **PLAYER** skin element. For more information, see Windows Media Player Skins.

See Also

- **Player Object**
- **Player Object Model**

About the PlayerApplication Object

The **PlayerApplication** object is used for remoting the Player. It provides the functionality required to switch between a remoted Player control and the full mode of the Player.
Remoting enables a Player control embedded in a C++ application to use the same playback engine as Windows Media Player. This means that usually you will use the PlayerApplication object in C++ code using the COM interfaces. There is a special case, however, where you can embed the Player control that displays a Windows Media Player skin as its user interface. In this case, PlayerApplication can be programmed using JScript in the skin code.

See Also

- Player Object Model
- PlayerApplication Object
About the Settings Object

The **Settings** object governs the settings of the control such as volume, play count, mute, and so on. It is accessed only through the **Settings** property of the **Player** object. The **Settings** property returns the **Settings** object. You can only access the properties of the **Settings** object after you have created it. For example, to get the value of the **Volume** property, you must use the following code:

```javascript
myvolume = player.settings.volume;
```

**See Also**

- **Player Object Model**
- **Settings Object**

About the StringCollection Object

The **StringCollection** object governs string collections, which are used to provide string handling capabilities for the **MediaCollection** object. The **StringCollection** object can be retrieved through the **getAttributeStringCollection** method of the **MediaCollection** object. You can use this to work with string collections representing the values of various media item attributes.

**See Also**
About the Object Model Versions

Windows Media Player 7.0 introduced a new object model. This object model has been extended with Windows Media Player 7.1, Windows Media Player for Windows XP, and Windows Media Player 9 Series. Each topic in the Object Model Reference includes a Requirements section that details the minimum requirement for the individual property, method, or event. The following lists detail the new objects, methods, properties, and events that have been added for each version since version 7.0.

Added for Windows Media Player 7.1

- **Player.stretchToFit** Property

Added for Windows Media Player for Windows XP

- **Controls.step** Method
- **DVD Object**
- **Media.error** Property
- **Player.DomainChange** Event
- **Player.MediaError** Event
- **Player.OpenPlaylistSwitch** Event
- **Player.windowlessVideo** Property

Added for Windows Media Player 9 Series

- **ClosedCaption.getSAMILangID** Method
- **ClosedCaption.getSAMILangName** Method
- **ClosedCaption.getSAMISyleName** Method
- **ClosedCaption.SAMILangCount** Property
- **ClosedCaption.SAMISyleCount** Property
- **Controls.audioLanguageCount** Property
- **Controls.currentAudioLanguage** Property
- **Controls.currentAudioLanguageIndex** Property
- **Controls.currentPositionTimecode** Property
- **Controls.getAudioLanguageDescription** Method
- **Controls.getAudioLanguageID** Method
- **Controls.getLanguageName** Method
- **ErrorItem.condition** Property

© 2000-2003 Microsoft Corporation. All rights reserved.
Properties, Methods, and Events

Each object has methods and properties through which you can program the Windows Media Player control. A method is an action that the object can take. A property is a data value that you can read or change. For example, the Play method starts the content playing, and the frameRate property indicates the current frame rate of the content that is playing.
In addition, the **Player** object raises events that give you the opportunity to carry out actions at specific times. You write code in an event handler that will execute when Windows Media Player raises the corresponding event. For example, you can write code in a **PlayStateChange** event handler that determines whether the change in state is that the media ended and if so display a dialog box asking users if they want to play the media again.

**Note**  All of the methods in the Player object model are asynchronous. If you call two methods in the same procedure, the second method cannot rely on the first method having completed its action.

**See Also**

- [About the Player Object Model](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Supported Protocols and File Types**

Windows Media Player supports several protocols and many file types, all of which can be used when specifying URL values in the Player object model for properties such as **Player.URL** or in Windows Media metafile playlists. Additionally, the supported protocols can be used when specifying protocol values with the proxy-related methods of the **Network** object.

The following digital media file formats are currently supported by Windows Media Player:

- Advanced Systems Format (ASF)
- AIF
- AIFC
- AIFF
- AU
- AVI
- MID
- MPE
- MPEG
- MPG
- MP2
- MP3
- M1V
- SND
- WAV
- Windows Media files with a .wm file name extension
- Windows Media Audio (WMA)
Windows Media Video (WMV)

The following protocols are currently supported by Windows Media Player.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol. Includes HTTP with fast cache and multicast.</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real Time Streaming Protocol. Includes RTSP with fast cache.</td>
</tr>
<tr>
<td>RTSPU</td>
<td>RTSP used with User Datagram Protocol (UDP). Includes RTSPU with fast cache</td>
</tr>
<tr>
<td>RTSPT</td>
<td>RTSP used with Transmission Control Protocol (TCP). Includes RTSPT with fast cache</td>
</tr>
<tr>
<td>MMS</td>
<td>Microsoft Media Server protocol.</td>
</tr>
<tr>
<td>MMSU</td>
<td>MMS used with UDP.</td>
</tr>
<tr>
<td>MMST</td>
<td>MMS used with TCP.</td>
</tr>
<tr>
<td>WMPCD</td>
<td>A protocol used by Windows Media Player to provide access to compact discs.</td>
</tr>
<tr>
<td>WMPDVD</td>
<td>A protocol used by Windows Media Player to provide access to DVD-ROM discs.</td>
</tr>
</tbody>
</table>

See Also

- About the Player Object Model
- WMPCD Protocol
- WMPDVD Protocol

© 2000-2003 Microsoft Corporation. All rights reserved.

WMPCD Protocol

The WMPCD protocol enables you to specify tracks on a compact disc using URL syntax. This is the general syntax of the protocol:

`wmpcd://drive/track`
The drive segment is the index of the CD drive. The track segment is the number of the track. The following example demonstrates using the WMPCD protocol.

```
player.url = "wmpcd://0/4";
```

The example plays the fourth track on the disc in the first CD drive (the drive whose index is 0).

See Also

- Supported Protocols and File Types

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

## WMPDVD Protocol

The WMPDVD protocol enables you to specify media from a DVD using URL syntax. This is the general form of the protocol:

```
wmpdvd://drive/title/chapter?contentdir=path
```

The drive segment is the letter of the DVD drive; it does not include the colon typically used with drive specifiers. This segment is always required.

The title segment is the number of the title to play. It is not required unless you want to start playback at a specific title or at a specific chapter of a specific title.

The chapter segment is the number of the chapter to play. It is not required unless you want to start playback at a specific chapter of a specific title.

The contentdir argument is the path to a VIDEO_TS.IFO file, which may be in local storage or on a network share. If you include this segment, then the drive segment is ignored although it is still required. If you also include the title segment or both the title and chapter segments then they are relative to the DVD content specified in the contentdir segment, not the drive segment.

The following example uses the WMPDVD protocol to play the DVD from the beginning, as if it were starting automatically.

```
player.url = "wmpdvd://F";
```

The following example uses the WMPDVD protocol to play the DVD from the beginning of the specified title.

```
player.url = "wmpdvd://F/2";
```
The following example uses the WMPDVD protocol to play the DVD from the specified chapter.

```javascript
player.url = "wmpdvd://F/2/4";
```

The following example uses the WMPDVD protocol to play DVD content from local storage. The `path` string ends with the folder that contains the VIDEO_TS.IFO file; it does not include the file name. In this example, the value of the `drive` segment has no effect although it is required, and playback begins in chapter 4 of title 2.

```javascript
player.url = "wmpdvd://Z/2/4?contentdir="d:\sample1\video_ts";
```

Assigning a WMPDVD string to the `url` property requires Windows Media Player 9 Series or later.

**See Also**

- Supported Protocols and File Types

---

**About Media Library**

**Media Library** is a database of information about the media content that is available to Windows Media Player on the user's computer. Some of the information is displayed in the **Media Library** pane in the player; all of it is available through code.

**See Also**

- About the Player Object Model
- Working with Media Library
Using HTML with Windows Media Player

Overview

Using HTML with Windows Media Player is an excellent way to combine audio and video with text and graphics. You can embed the Player control in a Web page when you want to supplement your static content or create Web applications with digital media. When you want to supplement your digital media with HTML, on the other hand, you can display Web pages in the full mode of the Player by referencing them in Windows Media metafile playlists.

If you write custom programs that embed the Player control in remote mode, you can also control the Web pages displayed in the various panes of the full mode of the Player when your users undock the control. This lets you preserve continuity between the docked and undocked states.

Web Embedding

When you embed the Player control in a Web page, you can take a basic approach or a more elaborate approach. In the basic approach, you use the control simply to play background music or to add a video display to your page. In the more elaborate approach, you make your page interactive by adding script code to manipulate the control through the Player object model. This lets you provide a user interface of your own design or respond dynamically to script commands or events such as media transitions.

For more information, see Embedding the Player Control in a Web Page.

Script Commands and URL Flipping

Script commands are text/value pairs you can embed in your digital media files or streams. You might use custom script commands solely to trigger script code, while letting Windows Media Player handle other script commands automatically.

When you have several Web pages that accompany a digital media presentation, URL script commands can automatically change the page in one frame while the Player control continues playing media in another frame. This is called URL flipping, and is an excellent way to create a multimedia slideshow. Other automatically handled script commands let you switch playback to a different media file or stream, display captioning text, or trigger events such as ad insertions defined in a Windows Media metafile playlist.

For more information about URL flipping, see Creating Web-Based Presentations.

Rich Media Streaming

URL flipping works best with simple pages that load rapidly. With more complex pages, multiple components are transferred individually, making it difficult to synchronize page display with digital media. To allow complex rich media presentations, Web pages can be added to a media stream and delivered to the Player in the same way as audio and video. This lets you synchronize the components of your presentation much more easily, especially over low-speed connections.

For more information about rich media streaming, see Creating Web-Based Presentations.

Browser Support

You can embed the Windows Media Player control in both Microsoft Internet Explorer and in Netscape Navigator, although the process is slightly different for each. You can also create Web pages designed to work
with both browsers.

With Internet Explorer, you embed the control using the HTML OBJECT element. Navigator requires a different approach, however, because it doesn't directly support ActiveX controls. With Navigator, you use the APPLET element to embed a special Java applet into the page. This applet handles communication with the Player ActiveX control.

For more information about Netscape Navigator support, see Using Windows Media Player with Netscape Navigator.

Displaying Web Pages in the Full Mode of the Player

You can extend the functionality of the Player or provide a custom view of information that accompanies your digital media by displaying Web pages in the full mode of the Player. This is a feature of Windows Media metafiles. Metafiles give you great control over playlist content, allowing you to seamlessly transition between clips, insert advertisements, and display still images in the Player. To display Web pages in the full mode of the Player, you use the PARAM element to add URL references to your playlist entries or to entire playlists.

For more information about using Web pages in metafiles, see Displaying Web Pages in Windows Media Player.

Displaying Web Pages in the Remoting Undocked State

When you embed the Player control in a custom program using remoting, you can let your users undock the control, switching to the full mode of the Player while media playback continues uninterrupted. This gives your users access to the features of the full mode of the Player, after which they can dock the Player again and return to your custom program.

While the Player is undocked, your program can specify the Web pages that display in the various panes of the Player such as the Now Playing and Media Guide panes. This lets you retain control of the user experience, including branding and advertising, while the Player is undocked.

For more information, see Remoting the Windows Media Player Control.

See Also

- About the Player Object Model
In addition to embedding the Windows Media Player control in a Web page as described in the previous section, you can embed it in Microsoft Office documents and in custom programs that you create using one of several programming languages, including Microsoft Visual Basic, C++, and C#.

The following sections provide information about using the control in various programming environments.

### Section Description

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding the Player Control in an Office Document</td>
<td>This section describes using the control in a Microsoft Office document.</td>
</tr>
<tr>
<td>Embedding the Player Control in a Custom Program</td>
<td>This section describes using the control in a Windows application written in Visual Basic or C++.</td>
</tr>
</tbody>
</table>

**See Also**

- [About the Player Object Model](#)

---

**Embedding the Player Control in an Office Document**

Embedding the Player control in an Office document is an easy way to add dynamic, interactive digital media content to an otherwise static document. For example, you can create a spreadsheet in Microsoft Excel and insert a "talking head" video summarizing a report, or you can create a Microsoft Word document and insert a short animation illustrating a point made in the text. If you don't like the user interface provided by the control, you can use Microsoft Visual Basic for Applications (VBA) to provide a custom user interface.

Embedding the Player control in a Microsoft PowerPoint® slideshow lets you supplement the dynamic effects already provided by that program. When you save the presentation as a Web page, the Player control is embedded in the HTML as described in [Embedding the Player Control in a Web Page](#). This lets you add additional script code that you may have developed for other Web pages.

Another way to easily generate Web pages that embed the Player control is with Microsoft FrontPage®. FrontPage lets you rapidly create a page, add the Player control from a menu, and configure it in a properties dialog box. Again, the resulting HTML embeds the Player control just as described previously.

**See Also**

- [Embedding the Player Control](#)
Embedding the Player Control in a Custom Program

Because the Windows Media Player ActiveX control is based on Microsoft Component Object Model (COM) technology, you can embed it in programs written with many different programming languages. The Player control represents an easy way to add sophisticated digital media functionality to any program.

In Microsoft Visual Basic, you can add the control to the control toolbox, place it on a form, and adjust the control properties in the properties window. If you want a custom user interface, you can place command buttons on the form and add code that manages the Player control. Embedding the control in a Visual Basic-based program is very similar to embedding it in an Office document and programming it with VBA.

The Microsoft Foundation Class (MFC) Library lets you embed the Player control with similar ease in C++ programs. Alternately, you can embed the control manually using COM methods to instantiate the control and access the COM interfaces documented in Object Model Reference for C++.

When you embed the Player control in a C++ program, you have the option of implementing COM interfaces that allow the control to run in remote mode. This means that the embedded control shares the same playback engine as the full mode of the Player, and users can switch back and forth between the full mode and the docked state without interrupting digital media playback. You can also control what is displayed in the various panes of the full mode Player when your users switch to the undocked state.

With C++ embedding, you also have the option of applying a skin definition file to the embedded Player control. This is an easy way to create lightweight user interface code that you can maintain separately from your main program code.

See Also

- Embedding the Player Control
- Embedding the Player Control in a C++ Program
Player Control Guide

The Microsoft Windows Media Player ActiveX control object model exposes objects, methods, properties, and events that you can use in skins and when you embed the control in Web pages or custom programs.

The following topics describe several ways to use the embedded Windows Media Player control.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding the Player Control in a Web Page</td>
<td>This topic describes how to use the Player control with Microsoft Internet Explorer and with Netscape Navigator.</td>
</tr>
<tr>
<td>Creating Web-Based Presentations</td>
<td>This topic describes how to use the Player control for URL flipping.</td>
</tr>
<tr>
<td>Adding Closed Captions to Digital Media</td>
<td>This topic describes how to use Synchronized Accessible Media Interchange (SAMI) format to supplement your digital media with captions.</td>
</tr>
<tr>
<td>Working with Media Library</td>
<td>This topic describes how to work with media items and playlists and their associated attributes.</td>
</tr>
<tr>
<td>Using Windows Media Player with Microsoft Office</td>
<td>This topic describes how to embed the Player control in Microsoft Office documents.</td>
</tr>
<tr>
<td>Using Windows Media Player with Visual Basic</td>
<td>This topic describes how to embed the Player control in applications created with Visual Basic 6.0.</td>
</tr>
<tr>
<td>Embedding the Player Control in a C++ Program</td>
<td>This topic describes several ways to embed the Player control in a custom C++ program.</td>
</tr>
<tr>
<td>Embedding the Player Control in a .NET Framework Solution</td>
<td>This topic describes how to embed the Player control in a .NET Framework solution.</td>
</tr>
<tr>
<td>Object Model Migration Guide</td>
<td>This topic describes how to upgrade pages designed for Windows Media Player 6.4 to make them work with the 9 Series object model.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player Object Model
Embedding the Player Control in a Web Page

Embedding the Windows Media Player control in a Web page lets you completely customize the way the user interacts with the control. You can use the interface provided by the control, or you can hide it and display your own user interface. You can specify multiple Player control properties at the point where you embed the control, or you can set Player properties and call Player methods in script code.

The following sections describe the basics of embedding the control in a Web page.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiding the Player Control</td>
<td>Describes the options for hiding the Player control when you want to provide your own user interface.</td>
</tr>
<tr>
<td>PARAM Tags</td>
<td>Describes how to initialize the Player control when you embed it.</td>
</tr>
<tr>
<td>Simple Example of Scripting in a Web Page</td>
<td>Describes a simple, but complete, example of an embedded Player control with a custom user interface.</td>
</tr>
<tr>
<td>Using Windows Media Player with Netscape Navigator</td>
<td>Describes how to use the Player control with Netscape Navigator. It also describes how to make your pages compatible with both Netscape Navigator and Microsoft Internet Explorer.</td>
</tr>
</tbody>
</table>

See Also

- Player Control Guide

© 2000-2003 Microsoft Corporation. All rights reserved.
The Windows Media Player ActiveX object is embedded in a Web page using the OBJECT element. Unlike earlier versions, the OBJECT element that defines Windows Media Player must be placed in the BODY section of a Web page; that is, between the <BODY> and </BODY> tags. Placing the Windows Media Player ActiveX object in the HEAD section of a Web page in order to hide the user interface can produce unexpected results.

If you put the Windows Media Player ActiveX control in the BODY section of a Web page, the control user interface will be displayed. If you do not want it to be displayed, and wish to create your own user interface, set the height and width attributes of the OBJECT element to zero.

The control can also be made invisible by setting the Player.uiMode property to "invisible". This can be done using a PARAM tag as discussed in the next section. In this case, space is reserved for the control using height and width, but nothing is displayed in the reserved space until uiMode is changed to something other than "invisible".

See Also

- Embedding the Player Control in a Web Page

PARAM Tags

Windows Media Player uses the PARAM element to define specific startup conditions for the control. The PARAM element is embedded inside the OBJECT element.

For example, if you want to define whether the autoStart property is True, you would embed the PARAM element inside the OBJECT element.

```xml
<Object ID="Player" CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
  <param name="autoStart" value="True"/>
</object>
```

You can have as many PARAM tags in an OBJECT element as you want. PARAM has two attributes, name and value. Both attributes must be set.

The following values are supported for use with the name attribute of the PARAM element.

- autoStart
- balance
- baseURL
- captioningID
- `currentPosition`
- `currentMarker`
- `defaultFrame`
- `enableContextMenu`
- `enabled`
- `fullScreen`
- `invokeURLs`
- `mute`
- `playCount`
- `rate`
- `SAMIFileName`
- `SAMILang`
- `SAMIStyle`
- `stretchToFit`
- `uiMode`
- `URL`
- `volume`
- `windowlessVideo`

See the [Object Model Reference](#) for more details about the values for each name attribute.

**See Also**

- [Embedding the Player Control in a Web Page](#)

---

**Simple Example of Scripting in a Web Page**

You can easily embed the Windows Media Player control in an HTML file using any scripting language your browser recognizes. The following simple example uses Microsoft JScript to create a page that will play a file when you click on a button, and stop playing the file when you click on another button.

You can embed the Windows Media Player ActiveX control in a Web page using the following four steps:

1. Create the Web page.
2. Add the `OBJECT` tag.
3. Add a user interface. In this case, two buttons.
4. Add a few lines of code to respond when the user clicks on one of the buttons you have created.

**Creating the Web Page**
The first step is to create a valid HTML Web page. The following code is the minimum needed to create a blank but valid HTML page:

```html
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

**Adding the OBJECT Tag**

Once you have created a Web page, you need to add an OBJECT tag. This identifies the ActiveX control to the browser and sets up any initial definitions. You must place the OBJECT tag in the BODY of the code. If you place it in the BODY, the default user interface of Windows Media Player will be visible. If you want to create your own user interface, set the height and width attributes to 0 (zero). You can also set the `Player.uiMode` property to "invisible" when you want to hide the control, but still reserve space for it on the page. The following code is recommended when you provide a custom user interface:

```html
<OBJECT ID="Player" height="0" width="0"
  CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>
```

The following OBJECT tag attributes are required:

**ID**

The name that will be used by other parts of the code to identify and use the ActiveX control. You can choose any name you want, as long as it is a name that is not already used by HTML, HTML extensions, or the scripting language you are using. In this example, the name `Player` is used, but you could also call it `MyPlayer` or something else. Just pick a name that is unique to that Web page.

**CLASSID**

A very large hexadecimal number that is unique to the control. Only one control has this number and it is the Windows Media Player ActiveX control. To prevent typographical errors, you can copy and paste this number from the documentation. Versions of the Windows Media Player control prior to version 7.0 had a different CLASSID.

**Adding a User Interface**

HTML allows a vast wealth of user interface elements, allowing the user to interact with your Web page by clicking, pressing keys, and other user actions. Adding a few INPUT buttons is the easiest way to provide a quick user interface. The following code creates two buttons that can respond to the user. Clicking one button starts the media stream playing and the other button stops it:

```html
<INPUT TYPE="BUTTON" NAME="BtnPlay" VALUE="Play" OnClick="StartMeUp()">
<INPUT TYPE="BUTTON" NAME="BtnStop" VALUE="Stop" OnClick="ShutMeDown()">
```

The name of the button is used to identify the button to your code; the value is the label that will appear on the button, and the OnClick attribute identifies which part of your scripting code will be called when the button is clicked.

**Adding Scripting Code**
Scripting code adds interactivity to your page. Scripting code can respond to events, call methods, and change run-time properties. Extended scripts are enclosed in a SCRIPT tag set. The SCRIPT tag tells the browser where your scripting code is and identifies the scripting language. If you do not identify a language, the default language will be Microsoft JScript.

It is good authoring practice to enclose your script in HTML comment tags so browsers that do not support scripting do not render your code as text. Put the SCRIPT tag anywhere within the BODY of your HTML file and embed the comment-surrounded code within the opening and closing SCRIPT tags.

The following Microsoft JScript code example calls the Windows Media Player control and performs an appropriate action in response to the corresponding button click.

```html
<SCRIPT>
<!--
function StartMeUp ()
{
    Player.URL = "laure.wma";
}

function ShutMeDown ()
{
    Player.controls.stop();
}
-->
</SCRIPT>

The example function, StartMeUp, is called when the button marked Play is clicked, and the ShutMeDown function is called when the Stop button is clicked.

The code inside StartMeUp uses the URL property to define a path to the media. The media will start playing immediately.

The ShutMeDown code calls the stop method of the Controls object. Note that the Controls object is called through the controls property of the Player object, which has the ID value of "Player".

The following code shows a complete example.

```html
<HTML>
<HEAD>
</HEAD>
<BODY>
<OBJECT ID="Player" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>

<INPUT TYPE="BUTTON" NAME="BtnPlay" VALUE="Play" OnClick="StartMeUp()">
<INPUT TYPE="BUTTON" NAME="BtnStop" VALUE="Stop" OnClick="ShutMeDown()">

<SCRIPT>
<!--
function StartMeUp ()
{
    Player.URL = "laure.wma";
}

function ShutMeDown ()
{
Player.controls.stop();

Note that you must provide a valid URL to a valid file name in the URL property. In this case the assumption is that the file laure.wma is in the same directory as the HTML file.

See Also

- Embedding the Player Control in a Web Page

Using Windows Media Player with Netscape Navigator

Netscape Navigator does not directly support the embedding of ActiveX controls. To provide this support for the Windows Media Player control, the Player install program includes a special Java applet that acts as an intermediary and host window for the Player ActiveX control.

The Player control is supported with Navigator versions 4.7, 6.2, and 7.0. Navigator versions 6.2 and 7.0 also require the Java 2 Runtime Engine (J2RE) version 1.3.x.

For pages viewed with Navigator, you use this applet in your page instead of embedding the Player ActiveX control using the HTML OBJECT element. If your audience includes both Internet Explorer and Netscape Navigator users, you must provide script code in your page that determines the current browser and uses the appropriate embedding technique accordingly.

Note  All the Player control functionality available in Internet Explorer is also available in Navigator except for the Settings.defaultFrame property. In Navigator, each URL-type script command received displays the URL in a new browser window, regardless of the value of Settings.defaultFrame.

Note  The Java applet does not have any automatic access to Media Library. However, you can still request the access rights that you need for your application. For general information about access rights, see Media Library Access.

Users of both browsers must install Windows Media Player before viewing pages that embed the control. If Windows Media Player is installed after Netscape Navigator, the Player applet and an accompanying dynamic
link library (DLL) are added to the appropriate Netscape directories automatically. In this case, no further setup is required by your users in order to view your pages in Navigator. You should instruct users who have installed Windows Media Player before Navigator to run the Player installation program again after Navigator has been installed.

The following topics provide additional information about using the Player control in Netscape Navigator:

- Adding the Player Applet to a Web Page
- Adding Script Code
- Handling Events in Netscape Navigator

See Also

- Embedding the Player Control in a Web Page

Adding the Player Applet to a Web Page

To embed the Player control in a Web page viewed with Netscape Navigator, you use the APPLET element instead of the OBJECT element used in Internet Explorer. For comparison, both techniques are shown below.

Web pages viewed with Internet Explorer use the following code to embed the Player control:

```html
<Object ID="Player" ClassID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6"></Object>
```

Web pages viewed with Netscape Navigator use this code:

```html
<Applet Name="Player" Code="WMPNS.WMP">
</Applet>
```

In both cases, you provide initial display settings using attributes such as height and width and initial Player property settings using the PARAM element. The mayscript attribute, which enables event handling, is described later. The following code shows an embedded applet with the URL property set:

```html
<Applet Name="Player" Code="WMPNS.WMP" Height="200" Width="200" Mayscript> 
  <param Name="URL" Value="video.wmv"/>
</Applet>
```

Web pages intended for both browsers use code that chooses the appropriate embedding technique depending on which browser the page is viewed with. In this case, put code such as the following SCRIPT element in the BODY element of your page at the location the Player window is to appear.
Adding Script Code

Once you have embedded the Player control, you can manipulate it in script code through the Player object. Your script code accesses this object through the name you provide when you embed the Player control. The OBJECT element uses the ID attribute to name the control while the APPLET element uses the NAME attribute.

With Netscape Navigator, you must access this root object name through the global document object. With Internet Explorer, you can use the document object or you can use the specified object name directly. The example code in this documentation uses Player for the control name and accesses it using document.Player in both cases.

You can use the Player object to call various methods and to specify or retrieve various properties. Some of these properties are child objects with additional methods and properties. For more information about the objects available and their methods and properties, see the Object Model Reference.

With Netscape Navigator, the object model properties are specified and retrieved using special methods called accessor methods. Accessor method names start with "set" or "get" and end with the name of a property. Read-only properties have a get accessor method while read/write properties have both set and get accessor methods. For example, the Player.uiMode property is specified for Netscape Navigator by using the setUiMode method and is retrieved by using the getUiMode method.

The set accessor methods take the value being specified as their only parameter, and do not return a value. The get accessor methods take no parameters, and return the value being retrieved.

The following example illustrates accessor methods using a Play button. This button, when clicked, calls the
StartMeUp function, which specifies the value of the Player.URL property, retrieves the Controls object through the Player.controls property, and calls the Controls.play method.

The following FORM element works in both Internet Explorer and Netscape Navigator:

```html
<FORM>
  <INPUT TYPE="BUTTON" VALUE="Play" ONCLICK="StartMeUp()"/>
</FORM>
```

Note that Navigator expects INPUT elements to be inside FORM elements, while in Internet Explorer, the FORM element is optional in cases where you handle the click event yourself. Including it in this example allows it to work with both browsers.

The StartMeUp function allows you to keep the browser-dependent code separate from the user interface. In Netscape, accessor methods are used to specify and retrieve the properties, while in Internet Explorer, the properties are manipulated directly:

```javascript
function StartMeUp() {
  if (navigator.appName == "Netscape") {
    document.Player.setURL("video.wmv");
    document.Player.getControls().play();
  } else {
    document.Player.URL = "video.wmv";
    document.Player.controls.play();
  }
}
```

See Also

- Using Windows Media Player with Netscape Navigator

Handling Events in Netscape Navigator

When you want your Navigator script code to handle events originating from the Player control, you must add the mayscript attribute to the APPLET element. This attribute does not require a value, but you can specify the value as "true".

The mayscript attribute tells the Player control to handle its events using functions that you provide. Each event handling function starts with "On" followed by an event signature as shown in the object model reference. For example, to handle the PlayStateChange event, write an OnPlayStateChange function such as the following:

```javascript
function OnPlayStateChange(NewState) {
  alert("Play state: " + NewState);
}
```
If you embed multiple Player controls in a single page, you can handle events from them individually by specifying a different EventPrefix applet parameter for each control. The value you specify is added to the name of the event handler function as shown in the following code:

```
<HTML>
<HEAD>
<SCRIPT>
function OnPlayer1PlayStateChange(NewState) {
    alert("Player1 play state: " + NewState);
}
function OnPlayer2PlayStateChange(NewState) {
    alert("Player2 play state: " + NewState);
}
</SCRIPT>
</HEAD>
<BODY>
<APPLET NAME="Player1" height="200" width="200"
 CODE="WMPNS.WMP" MAYSCRIPT>
<PARAM NAME="EventPrefix" VALUE="Player1"/>
<PARAM NAME="URL" VALUE="video1.wmv"/>
</APPLET>
<APPLET NAME="Player2" height="200" width="200"
 CODE="WMPNS.WMP" MAYSCRIPT>
<PARAM NAME="EventPrefix" VALUE="Player2"/>
<PARAM NAME="URL" VALUE="video2.wmv"/>
</APPLET>
</BODY>
</HTML>
```

Note that the example sets the EventPrefix and the applet NAME attribute to the same value. The EventPrefix is used solely with functions that handle Player control events, while the NAME value is used solely in script code that manipulates the control. Using the same value for both makes it easier to keep track of which functions relate to which instance of the control.

For pages viewed with both Navigator and Internet Explorer, you must provide event handlers for each. Event handlers for Internet Explorer are SCRIPT elements that specify both the event being handled and the control name that the event must originate from. This name corresponds to the ID attribute specified in the OBJECT element. The following code demonstrates event handlers for two embedded controls:

```
<SCRIPT language="JScript" for="Player1" event="PlayStateChange(NewState)">
    alert("Player1 play state: " + NewState);
</SCRIPT>
<SCRIPT language="JScript" for="Player2" event="PlayStateChange(NewState)">
    alert("Player2 play state: " + NewState);
</SCRIPT>
```

Because Internet Explorer and Netscape Navigator each use one event handling technique and ignore the other, you can use both techniques in a single page without wrapping them in browser detection code.

**See Also**

- [Using Windows Media Player with Netscape Navigator](#)
Creating Web-Based Presentations

The Windows Media Player control lets you easily create Web-based slideshow presentations that combine audio and video with HTML. By adding URL-type script commands to your media files, you can cause specified Web pages to display in a specified Web browser frame at specific times during digital media playback.

Windows Media Player also features rich media streaming to enable efficient delivery of HTML data over a network inside a single Windows Media stream or file. The Player and server handle the smooth, timely delivery of audio, video, and HTML to the browser. Just as in Web presentations that do not use rich media streaming, embedded script commands render the presentation in a specified Browser frame at specified times.

A typical layout for Web-based presentations uses two frames. One of the frames contains a Web page that embeds the Player control and displays the video part of a presentation. The other frame displays Web pages that change at various times as the video plays.

If the digital media accompanying your Web pages is audio only, you can hide the frame that contains the Player control page by setting its width to zero. This way, the audio can play uninterrupted in the background while your Web pages display in the full browser window.

The following sections describe common techniques for enabling Web-based presentations:

- [URL Flipping](#)
- [Rich Media Streaming](#)

See Also

- [Player Control Guide](#)
URL Flipping

Using Web pages for slideshows is called URL flipping, and is handled automatically by Windows Media Player when it encounters URL-type script commands embedded in the digital media stream it is playing. You can specify a target frame for your pages in each script command, or you can specify it by setting the `Settings.defaultFrame` property. You can set this property in script code or by using a PARAM element within the OBJECT element that embeds the Player control.

You are free to handle the URL-type script commands in your JScript code just as you would handle custom script commands. If you want the Player control to ignore URL-type script commands so that you can handle them entirely by yourself, set the `Settings.invokeURLs` property to false either in script code or using a PARAM element as previously described.

The following example illustrates a typical frameset for URL flipping. First, create a page that describes the frameset:

```html
<HTML>
<FRAMESET cols="300, *">
  <FRAME name="player" src="player.html">
  <FRAME name="content">
</FRAMESET>
</HTML>
```

Next, create the player.html file referenced in the frameset using the following code (the video.wmv file is assumed to exist in the same directory as the HTML files):

```html
<HTML>
<BODY>
<OBJECT ID="Player" CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
  <PARAM name="URL" value="http://www.proseware.com/Media/video.wmv"/>
  <PARAM name="defaultFrame" value="content"/>
</OBJECT>
</BODY>
</HTML>
```

As you can see, the basic setup is quite simple. If your Web pages are small enough to load rapidly, this setup may be sufficient for your needs. If, on the other hand, your Web pages are complex, rich media streaming may be more effective depending on the connection speed of your audience.

**Note** URL flipping does not work correctly with pages viewed in Netscape Navigator. Instead of appearing in the frame specified by `defaultFrame`, each URL-type script command received in Navigator displays the URL in a new browser window.

See Also

- [Creating Web-Based Presentations](#)
- [Using Script to Control URL Flipping](#)
Rich Media Streaming

Complex Web pages contain many different components that are normally transferred separately. On a slow connection, such pages display one piece at a time and may take several minutes to display completely. This may prevent you from effectively synchronizing your pages with your digital media. The solution to this problem is rich media streaming, which means adding your Web pages to your digital media stream so that they are delivered at the same time as the audio or video data.

Rich media streaming uses the same frameset layout described above and behaves exactly like normal URL flipping. Just as in URL flipping, URL-type script commands embedded in rich media streams are used to display the content in the target frame. Instead of pointing to pages on the Web, however, rich media stream URLs are used by Windows Media Player to access files in the Internet Explorer cache where the streamed HTML content is placed as it is received. Just as with normal URL flipping, you are also free to write event handlers that respond to these URL-type script commands, or you can let the Windows Media Player control handle everything automatically.

Note Any HTML content delivered through rich media streaming is played back in the Internet security zone, and is subject to the end user's browser security settings.

For more information on creating rich media presentations, see the Windows Media Format Software Development Kit (SDK) and the Windows Media Encoder SDK.

See Also

- Creating Web-Based Presentations
- Using Script to Control URL Flipping

Using Script to Control URL Flipping

When a user connects to a rich media stream while the stream is already in progress, it is possible for the streamed Web page to display before all the elements have arrived and been cached if Windows Media Player automatically invokes the URL. When this happens, the user sees a blank or incomplete Web page until the next set of data arrives in the cache. While this is normal behavior under these circumstances, you can avoid displaying a blank or incomplete Web page by invoking the URL using script instead of letting the Player do it
automatically. That way, you can ignore the first URL flip and then invoke subsequent URLs using script code.

**Note** This section assumes that you are streaming HTML using the Windows Media Encoder 9 Series SDK and that you have set the HTML stream to repeat.

First, you must create a frameset Web page to contain the frame with the embedded Player and the frame that displays the streaming HTML. Each of these two frames will display a separate Web page initially, so you will create a total of three Web pages. The following example code demonstrates the frameset Web page:

```html
<HTML>
<HEAD>
</HEAD>

<FRAMESET cols = "350, *">
    <FRAME  name = "player" src = "embed_player.htm">
    <FRAME  name = "content" src = "blank.htm">

    <NOFRAMES>
        <BODY>
            <P>This page uses frames, but your browser doesn't support them.</P>
        </BODY>
    </NOFRAMES>
</FRAMESET>
</HTML>
```

The preceding Web page example incorporates two frames. The first frame displays in the left half of the browser window and displays the Web page named `embed_player.htm`. The following example code creates this Web page:

```html
<HTML>
<HEAD>
</HEAD>

<!-- Embed Windows Media Player and disable the invokeURLs parameter -->
<object classid = "CLSID:6BF52A52-394A-11D3-B153-00C04F79FAA6" id = "Player">
    <param name = "URL" value = "http://www.proseware.com/Media/video.wmv">
    <param name = "invokeURLs" value = "false">
</object>

<script language = "JScript">
    var bFirstURL = true; // Global flag for first URL flip.
</script>

<!-- Create an event handler for script commands. -->
<script language = "JScript" for = "Player" event = "ScriptCommand(scType, scParam)">
    if("URL" == scType)
    {
        if ( bFirstURL == false )
        {
            // Show the next URL flip.
            parent.content.location = scParam;
        }
        else
        {
            bFirstURL = false; // Set the flag.
        }
    }
</script>
```
The second frame in the frameset displays in the right half of the browser window and displays a Web page named "blank.htm". The following example code creates this Web page:

```html
<HTML>
<HEAD>
</HEAD>
<BODY>
  Loading...
</BODY>
</HTML>
```

When the frameset page loads in the browser, the left frame shows the embedded Player and the right frame shows the text "Loading..." to inform the user that more data is forthcoming. When the first URL script command arrives from the HTML stream, the event handler simply changes the value of the `Boolean` flag. When each subsequent URL script command arrives from the HTML stream, the script in the event handler loads the new URL into the frame named "content", and the complete Web page displays in the frame located in the right half of the browser window.

For more information about streaming HTML using Windows Media, see the Windows Media Encoder 9 Series SDK.

**See Also**

- [Rich Media Streaming](#)
- [URL Flipping](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Adding Closed Captions to Digital Media**

Synchronized Accessible Media Interchange (SAMI) is a file format designed to deliver synchronized text such as captions, subtitles, or audio descriptions with digital media content. Integrated in a Web browser using the Windows Media Player object model, SAMI promotes accessibility. By using SAMI, you can create rich media content that reaches a larger, more diverse audience.
In addition to standard fonts, SAMI can support other text styles, such as different colors, sizes, or languages to aid a variety of users. SAMI can be particularly useful for individuals who have low vision or those who are deaf or hard-of-hearing. The SAMI format can also assist in educational purposes, such as teaching students a second language.

This topic focuses on the incorporation of SAMI with the Windows Media Player ActiveX control object model. SAMI files exist independently from digital media files and do not rely on a specific video or audio format to function. Since the files are separate, the Player control will locate, parse, synchronize, and render each file on the client's computer. This provides for added flexibility and functionality because it allows for the editing of individual SAMI files, the incorporation of the SAMI file with different digital media formats, and the storage of SAMI files on different server locations.

This topic contains the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About SAMI Files</td>
<td>Describes the basic structure of SAMI files.</td>
</tr>
<tr>
<td>SAMI File Example</td>
<td>Describes the structure of an example SAMI file.</td>
</tr>
<tr>
<td>Using SAMI with the Windows Media Player Control in a Browser</td>
<td>Describes how to use SAMI in a Web page with the Windows Media Player control.</td>
</tr>
<tr>
<td>About SAMI URLs</td>
<td>Describes how digital media files can be associated with SAMI files by using a single URL.</td>
</tr>
</tbody>
</table>

See Also

- Player Control Guide

About SAMI Files

SAMI files are text files that have an .smi or .sami file name extension. They contain the text strings used for synchronized closed captions, subtitles, and audio descriptions. They also specify the timing parameters used by the Windows Media Player control to synchronize closed caption text with audio or video content. When a digital media file reaches a time designated in the SAMI file, the text changes accordingly in the closed caption display area of the Web page.

Other than a simple text editor (such as Microsoft Notepad, Microsoft Word, or Apple SimpleText), special software is not required to create a SAMI file. Moreover, an extensive programming background is not needed.
to create SAMI files. In fact, if you are already familiar with Hypertext Markup Language (HTML), you will notice many similarities in flexibility, layout, and syntax. SAMI and HTML share common elements, such as the <HEAD> and <BODY> tags. Tags must always be used in pairs; for example, a BODY element begins with a <BODY> tag and must always end with a </BODY> tag.

A basic SAMI file requires three fundamental tags: <SAMI>, <HEAD>, and <BODY>. The <SAMI> tag identifies the document as a SAMI document so other applications can recognize its file format. Between the <HEAD> and </HEAD> tags, you define basic guidelines and other format information for the SAMI document, such as the document title, general information, and style properties for closed captions. Like HTML, content declared within the HEAD element does not display as output. Elements and attributes defined between the <BODY> and </BODY> tags display content seen by the user. In SAMI, the BODY element contains the parameters for synchronization and the text strings used for closed captions.

Defined within the HEAD element, the STYLE element provides for added functionality in SAMI. Between the <STYLE> and </STYLE> tags, you can define several Cascading Style Sheet (CSS) selectors for style and layout. Style properties such as fonts, sizes, and alignments can be customized to provide a rich user experience while also promoting accessibility. For example, defining a large text font style class can improve the readability for users who have difficulty reading small text. In addition, by defining several different language classes, you can help international users better understand the digital media content.

See Also

- [Adding Closed Captions to Digital Media](#)

SAMI File Example

The following example code is a complete SAMI file with one set of closed caption text and several class declarations for text style and caption language.

```xml
<SAMI>
<HEAD>
  <!-- P defines the basic style selector for closed caption paragraph text */
  P {font-family: sans-serif; color:white;}
  /* Source, Small, and Big define additional ID selectors for closed caption text */
  #Source {color: orange; font-family: arial; font-size: 12pt;}
  #Small {Name: SmallTxt; font-size: 8pt; color: yellow;}
  #Big {Name: BigTxt; font-size: 12pt; color: magenta;}
  /* ENUSCC and FRFRCC define language class selectors for closed caption text */
  .ENUSCC {Name: 'English Captions'; lang: en-US; SAMIType: CC;}
  .FRFRCC {Name: 'French Captions'; lang: fr-FR; SAMIType: CC;}
```
Using SAMI with the Windows Media Player Control in a Browser

By embedding the Windows Media Player control in a Web page, you can customize the user experience to a significant degree. Incorporating closed captions by this method provides the user with convenience, flexibility, and ease of management. Instead of creating a SAMI file for each font style or language, you can declare different style classes in one file by using basic scripting and the Player control object model. With the methods and properties available in the object model, you can create custom controls that enable the user to choose...
between the different style and language options. Furthermore, you have complete control over the design of the Player interface and the customization of each function.

For detailed information about embedding the Windows Media Player control in a Web page, see Simple Example of Scripting in a Web Page.

The following example code demonstrates how to use closed captions with the Windows Media Player control embedded in a Web page. It includes controls to allow the user to select font style and language.

<HTML>
<HEAD>

<SCRIPT>
   // The following variable is used to prevent multiple initialization.
   var initialized = false;
   // The following function populates the select boxes.
   // It is called the first time the media file is opened.
   // Before then, the SAMI settings cannot be retrieved.
   function initialize() {
      var newOption;
      for (var i = 0; i < Player.closedCaption.SAMILangCount; i++) {
         newOption = document.createElement("OPTION");
         newOption.text = Player.closedCaption.getSAMILangName(i);
         newOption.value = newOption.text;
         CCLang.options.add(newOption);
      }
      for (var i = 0; i < Player.closedCaption.SAMIStyleCount; i++) {
         newOption = document.createElement("OPTION");
         newOption.text = Player.closedCaption.getSAMIStyleName(i);
         newOption.value = newOption.text;
         CCStyle.options.add(newOption);
      }
      initialized = true;
   }

   <!-- The following script code runs when the page is fully loaded. -->
   <SCRIPT for="window" event="onload()">
      Player.closedCaption.captioningID = "captions";
      Player.closedCaption.SAMIFileName = "http://www.proseware.com/Media/seattle.smi";
      // The digital media file will open automatically, after which
      // the OpenStateChange event (handled below) will fire.
      Player.URL = "http://www.proseware.com/Media/seattle.wmv";
   </SCRIPT>

   <!-- The following script code runs when a media file is opened. -->
   <SCRIPT for="Player" event="OpenStateChange(NewState)">
      // The first time this event fires, the Player stops and the
      // initialize function is called. This allows the user to
      // select a language and style before viewing the file.
      if (13 == NewState && !initialized) {
         Player.controls.stop();
         initialize();
      }
   </SCRIPT>

</HEAD>

<BODY>

<OBJECT
ID="Player"
About SAMI URLs

SAMI files can be associated with digital media files using a single URL. This is accomplished by using the `sami` URL parameter. The URL parameter is preceded by the base URL and a `?` character. A URL with a `sami` parameter follows this syntax: `URL?sami=captionsURL`. The value of the URL parameter follows the parameter name and an equals sign, as in the following example:

```
```

This URL syntax is commonly used in a hyperlink or a Windows Media metafile to link directly to the locations of both the digital media file and the SAMI file. When the user clicks on the hyperlink, the Player launches in full mode and plays the digital media content.

If the `sami` URL parameter is not specified, Windows Media Player will look for a SAMI file in the same location as the digital media file and with the same file name except for the file name extension, which must be `.smi`. If such a file is present, it will be opened automatically if caption display has been enabled in the
Closed captions are enabled in the Player by clicking the **Play** menu, then clicking **Captions and Subtitles**, and then clicking **On**. If closed captions are enabled, the captions contained in the SAMI file will display while the digital media plays.

**See Also**

- [Adding Closed Captions to Digital Media](#)

---

**Working with Media Library**

**Media Library** is a database of information about the media content that is stored on the user's computer or has been played there. Some of the information is displayed in the **Media Library** feature in Windows Media Player; a larger set of information can be accessed programmatically.

Media items can be organized in playlists. Playlists can be created by users, and you can create them through code. The Windows Media Player object model provides objects and methods you can use to manipulate both individual media items and playlists.

The following sections cover the techniques for working with **Media Library**:

- [Media Library Access](#)
- [Managing Media Items](#)
- [Media Item Attributes](#)
- [Managing Playlists](#)

**See Also**

- [Player Control Guide](#)
Media Library Access

Properties and methods of the Windows Media Player object model that access Media Library require either read-only or read/write access to the database. Media Library contains information that some users want to keep private and that should only be accessed or altered with their consent.

To determine the current level of access granted to your code, retrieve the Settings.mediaAccessRights property. That property returns "none", "read", or "full" (read/write). To request specific access rights, call the Settings.requestMediaAccessRights method, passing a parameter that specifies the level you are requesting. The method displays a message to the user explaining the requested level of access, and returns a Boolean value indicating whether the access was granted.

Certain access rights are granted automatically depending on where your code is running relative to the user's computer.

- If your Web page or program is located on the user's computer, full access rights are granted by default.
- Web pages have read access to Player.currentMedia, Player.currentPlaylist, and Media.sourceURL when the Web page is located in an Internet Explorer security zone that is the same as or less restricted than the security zone of the media item or playlist.

Ranging from least restricted to most restricted, the security zones are the Trusted zone (including the user's local computer), the Local intranet zone, the Internet zone, and the Restricted zone.

For example, a Web page in the Local intranet zone has full access rights to Player.currentMedia when the corresponding media item is on the local intranet or the Internet, but access rights must be requested for media items located on a user's local computer or on a Web site in the Trusted zone.

- For Netscape Navigator, the Java applet does not have any automatic access to Media Library. However, you can request the access rights you need for your application. For general information, see Using Windows Media Player with Netscape Navigator.

You should test your Web-based or Windows-based application in all of the security zones it may encounter. The application should be designed to handle denial of an access request correctly.

Windows Media Player object model versions prior to Windows Media Player 9 Series do not include mediaAccessRights or requestMediaAccessRights.

See Also

- Settings Object
- Working with Media Library
Managing Media Items

A Media object represents one media item. It has properties and methods you can use to retrieve information and display it to the user, or to take different actions based on the value you retrieve.

Much of your work with Media objects involves metadata about the content of the media item, called the attributes. The topic Media Item Attributes describes how to read and change attribute values.

The Media object has properties and methods that retrieve some attributes directly, such as the name or duration of the item. For video items, you can retrieve the height and width of the image, and you can retrieve marker information based on the name or index of a marker. You can also determine whether a particular media item is included in a particular playlist.

Retrieving a Media Object

You can quickly access the current media item using the Player.currentMedia property. You can also access that same item using the Controls.currentItem property.

The following JScript examples each retrieve the same value, a Media object representing the current item. (Throughout this topic, the player object was created with ID="player").

```javascript
var oMedia1 = player.currentMedia;
var oMedia2 = player.controls.currentItem;
```

You can create a new media item from a media file using the Player.newMedia method. You pass the method the URL path to a media file, and it returns a reference to the new Media object. The method does not add the new object to Media Library directly; however, you can pass the object to the Playlist appendedItem method or the Playlist.insertItem method.

The following JScript example creates a Media object based on one of the SDK media sample files.

```javascript
var oMedia3 =
    player.newMedia("C:\\WMSDK\\WMPSDK9\\samples\\media\\laure.wma");
```

Note You must include two backslash (\) characters in a string to represent one actual backslash character. This is because JScript uses a single backslash character to define an escape sequence.

You can create a new media item from a media file and add it to Media Library in one step by using the MediaCollection.add method. Like the Player.newMedia method, the add method takes a path to a digital media file.

The following JScript example creates a Media object based on one of the SDK media sample files and adds that object to Media Library.

```javascript
var oMedia4 = player.mediaCollection.add(
```
You can retrieve a Media object representing a media item in a playlist using the Playlist.item method. The following JScript example retrieves the sixth media item from the current playlist.

```javascript
var oMedia5 = player.currentPlaylist.item(5);
```

See Also

- Controls.currentItem
- Managing Playlists
- Media Object
- MediaCollection.add
- Player.currentMedia
- Player.newMedia
- Playlist.item
- Working with Media Library

Media Item Attributes

When you look at Media Library in Windows Media Player, you typically see a great deal of information about a media item. In addition to the name of an audio track, for example, you will likely see the name of the album the track is on, the names of the artist and composer, the genre of the music, and so on.

In the Windows Media Player object model, these metadata items are called attributes. The object model includes properties and methods you can use to query media items and playlists for attributes. You can then display attribute values in your user interface, or your code can take actions based on the value of an attribute.

The following topics describe working with media item attributes.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding Attribute Sources</td>
<td>Describes how the source of a media item affects the attributes that may be associated with it.</td>
</tr>
<tr>
<td>Reading Attribute Values</td>
<td>Shows the methods for retrieving the value of an attribute.</td>
</tr>
<tr>
<td>Attributes with Multiple Values</td>
<td>Shows the methods for retrieving the value of a multi-valued attribute.</td>
</tr>
</tbody>
</table>
The following topics list the attributes that may be associated with a media item and therefore may be available to your application.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Attributes</td>
<td>Lists the attributes that you can access with getItemInfo and other methods.</td>
</tr>
<tr>
<td>Multi-Valued Attributes</td>
<td>Lists the attributes that can have multiple values.</td>
</tr>
<tr>
<td>Attribute Aliases</td>
<td>Lists the attribute names that you can use that are actually stored with a different name.</td>
</tr>
<tr>
<td>Media File Attributes</td>
<td>Lists the attributes that are stored in the media item file, and the constants you would use with Windows Media Format SDK to work with them.</td>
</tr>
</tbody>
</table>

See Also

- Playlist Attributes
- Working with Media Library

Understanding Attribute Sources

The attributes, or metadata, that you can use come from a variety of sources. This topic describes those sources, moving from scenarios with fewer potential attributes to those with more potential attributes.

While the user plays a CD or DVD, you have access to very little metadata about the disc itself or the media content of the disc. Commercial discs do not typically include attribute metadata.

If the user has played a CD or DVD while they were connected to the Internet, you may have access to more attributes when that disc is in the disk drive. While Windows Media Player is connected to the Internet and playing a CD or DVD, the Player retrieves metadata for that disc from an online database. The Player displays this information in Now Playing and in the Now Playing category in Media Library. The attributes are not stored in the Media Library database, but they are cached. If the cache has not yet been flushed, your...
application will have access to the attributes while the disc is in the drive.

**Note** Users may choose to disallow retrieving media information from an online database. In that case, the only attributes available to you will be those from a media file, those that the user has manually entered in **Media Library**, and those generated by the Player itself (such as the attributes related to how frequently an item has been played).

If the user plays a digital media file that is not added to **Media Library**, you have access to the attributes that are in the file.

If the user plays a digital media file that has been added to **Media Library**, you have access to the attributes that are stored only in **Media Library**, the attributes that are stored only in the file, and the attributes that are stored in both **Media Library** and the file.

The attributes that are available for media items added to **Media Library** vary according to how the source digital media file was created and what actions the user has taken since adding it.

- The content creator may insert attribute information in the file when it is first created. For example, if you create and distribute a digital media file with your application, then you have control over which attributes are originally inserted in the file.
- If the user modifies attribute data for a media item that has been added to **Media Library**, using Advanced Tag Editor or the **Media Library** user interface, Windows Media Player adds that data to the **Media Library** database. It adds some attributes directly to the file because they are stored only in the file; those attributes are identified in [Media File Attributes](#). At some indeterminate time, the **Media Library** database synchronizes with the file, so that attributes that are stored in both **Media Library** and the file have the same value.
- If the user copies a track from a CD by using Windows Media Player while connected to the Internet, the effect is nearly the same as if the user had modified attributes by using Windows Media Player. Attributes are added to the **Media Library** database, drawn from the file itself and from an online database. Some attributes are stored only in the file. At some indeterminate time the **Media Library** database synchronizes with the file.
- If you write code using the Player control to change the value of an existing attribute in a media item that has been added to **Media Library**, the effect is nearly the same as if the user had modified the attribute using Windows Media Player. The value is written to the **Media Library** database and at some indeterminate time the database synchronizes with the file.

**Note** If you embed the control in your application, file attributes that you change will not be written to the media file itself until the user runs Windows Media Player. If you use the control in a remoted application written in C++, file attributes that you change will be written to the media file itself shortly after you make the changes. In either case, the changes are immediately available to you through **Media Library**.

- If you write code using the Player control to insert a custom attribute into a media item, the attribute and its value will persist only as long as your application has a reference to the **Media** object. Neither the attribute nor its value will be stored in the **Media Library** database or the media file, if there is one.

The simplest situation is when you are working with digital media files that you have supplied. In that scenario, you know that specific attributes are in the file; when you add the media item to **Media Library**, you know that you can work with those attributes.

**See Also**

- [Available Attributes](#)
Reading Attribute Values

The attributes you can find in Media Library and in Windows Media files have pre-defined names. You can write code that retrieves the value of one attribute by passing the name of that attribute to Media.getItemInfo or Media.getItemInfoByType. You can also write code that retrieves the values of all of the attributes in a file or item.

The following JScript example retrieves the value of a particular attribute, in this case the Title attribute. In this example, the player instance was created with ID="player".

```javascript
var oMedia = player.currentMedia;
var attribValue = ";

// Retrieve the value of the attribute.
attribValue = oMedia.getItemInfoByType("Title", "", 0);
```

In the call to getItemInfoByType, the second parameter is a string that specifies the language. If you pass an empty string, as this example does, the method retrieves the value in the default language. For information about the third parameter, see Attributes with Multiple Values.

The following JScript example retrieves the values of all of the attributes in the current media item. In this example, the player instance was created with ID="player".

```javascript
var numAttribs = 0;
var attribName = "";
var attribValue = "";
var oMedia = player.currentMedia;

// Retrieve how many attributes are in the file.
numAttribs = oMedia.attributeCount;

// For each attribute, retrieve its name
// and its value.
for (var i = 0; i < numAttribs; i++){
    attribName = oMedia.getAttributeName(i);
    attribValue = oMedia.getItemInfoByType(attribName, "", 0);
}
```

The Media.attributeCount property returns the number of attributes that are actually present in the file. This is typically a small subset of the potential attributes. In addition, the number of attributes in the file may change over time; for example, the user may insert an attribute using the Advanced Tag Editor or the Media Library.
Some media item attributes can have multiple values. For example, the Author, WM/Composer, and WM/Genre attributes can each have more than one value. The data type of such attributes is multi-valued string.

In Windows Media Player, Media Library displays multiple values in a single field, separating the values with semicolons. However, each value is actually a separate attribute in the Windows Media item.

You can write code that will determine whether a given attribute has multiple values and then retrieve all of those values. You must use Media.getItemInfoByType. If you use the Media.getItemInfo method to retrieve a multi-valued attribute, you will only retrieve the first value.

The following JScript example uses getItemInfoByType to retrieve a multi-valued attribute. In this example, the player instance was created with ID="player" and the current media item is an audio track that has multiple genre values.

```javascript
var oMedia = player.currentMedia;
var numInstances = 0;
var thisGenre = ";
var multiGenre = ";

// Determine how many values there are for the attribute.
numInstances = oMedia.getAttributeCountByType("WM/Genre", ");

// Retrieve all of the values for the attribute.
for (var i = 0; i < numInstances; i++) {
    thisGenre = oMedia.getItemInfoByType("WM/Genre", ",", i);
    multiGenre += thisGenre + "; ";
}
```

In the call to getAttributeCountByType, the second parameter is a string that specifies the language. If you pass an empty string, as this example does, the method retrieves the value in the default language.
The third argument passed to the `getItemInfoByType` method is the index of a particular attribute in a set of attributes with the same name.

You can use similar code to retrieve attributes that have unique names. In those cases, `getAttributeCountByType` returns 1; in the example shown earlier, the call to `getItemInfoByType` would execute only once.

**See Also**

- Reading Attribute Values Sample
- Media Item Attributes
- Media Object

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Reading Attribute Values Sample**

The following JScript sample retrieves the values of all attributes in the current media item, including all values of multi-valued attributes. It combines the techniques described in the previous two sections.

You can copy this sample to a text file and save the file with the .htm file name extension. To work correctly, the media file Laure_multi.wma must be in the same folder as the HTML file. By default, the media file is installed at C:\WMSDK\WMPSDK9\samples\media. This WMA file has multiple values for the WM/Genre attribute.

```
<html>
<head>
<title>Retrieve Attributes Sample</title>
</head>
<body>
<table>
<tr>
<td width="50%"><center>
<object id="player"
classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
<param name="autoplay" value="true">
<param name="url" value="Laure_multi.wma">
</object><br><br>
<input type="button" id="btnShowAttribs" name="btnShowAttribs"
value="Show Attributes" onclick="showAttribs()"></center></td>
<td width="50%"><center>
<textarea id="txtAttribs" name="txtAttribs" cols="50" rows="20"></textarea></center></td>
</tr>
</table>
</body>
</html>
```
function showAttribs() {
    var numAttribs = 0;
    var attribName = "";
    var numInstances = 0;
    var attribValue = "";
    var oMedia = player.currentMedia;

    // Clear the TEXTAREA element first.
    txtAttribs.value = "";

    // Retrieve the number of attributes for the item.
    numAttribs = oMedia.attributeCount;

    for (var i = 0; i < numAttribs; i++){
        // Retrieve the name of this attribute.
        attribName = oMedia.getAttributeName(i);

        // Retrieve the number of values for this attribute.
        numInstances = oMedia.getAttributeCountByType(attribName, "");

        try {
            // Retrieve each value for this attribute.
            for (var j = 0; j < numInstances; ++j) {
                attribValue = oMedia.getItemInfoByType(attribName, "", j);
                txtAttribs.value += attribName + " -- " + attribValue + "\n";
            }
        } catch(err) {
            alert(err.description);
        }
    }
}
</SCRIPT>
</BODY>
</HTML>

See Also

- [Media Item Attributes](#)
- [Media Object](#)
You can change the value of an attribute if your Web page or application has read/write access to Media Library and the attribute can be both read and written.

You can only change an attribute of the current media item. To change attributes of multiple media items, you must assign each one in turn to the `Player.currentMedia` property.

To change an attribute, call the `Player.currentMedia.setItemInfo` method. The following JScript example does this. In this example, the player instance was created with ID="player".

```javascript
var oMedia = player.currentMedia;
var newValue = "New genre";
var gotFullAccess = false;

gotFullAccess = player.settings.requestMediaAccessRights("full");
if(gotFullAccess){
    try {
        oMedia.setItemInfo("WM/Genre", newValue);
    } catch(err) {
        alert(err.description);
    }
} else {
    alert("Not allowed to change the attribute value.");
}
```

You should always explicitly request full access, so that your application or Web page will work correctly when it runs from different security zones. This will also give the user the opportunity to determine whether to grant full access.

You can call the `Media.isReadOnlyItem` method to determine whether you can change a particular attribute, if you do not already know this.

**Note** If you embed the control in your application, file attributes that you change will not be written to the media file itself until the user runs Windows Media Player. If you use the control in a remoted application written in C++, file attributes that you change will be written to the media file itself shortly after you make the changes. In either case, the changes are immediately available to you through Media Library.

See Also

- Media Library Access
- Reading Attribute Values
- Media Item Attributes
- Media Object
### Available Attributes

The following table lists attributes that may be available for a given item. There may be additional attributes available as well, such as custom attributes defined by the author of the media item. The attributes available for a given item and the values of those attributes depend on how the item was authored and on actions the user may have taken in Media Library. See Understanding Attribute Sources for more information.

In the table, the first column shows the name that you use with methods such as `Media.getItemInfo` and `Media.getItemInfoByType`. The second column shows whether the attribute is read-only (RO) or can be both read and written (RW). The third column shows the types of Media Library items that may have the attribute. If this field is empty, the attribute is stored only in the file. The file-only attributes are also identified in the Media File Attributes topic. The values used in this column are the possible values of the MediaType attribute; for more information, see the MediaCollection.getByAttribute topic.

The fourth column shows whether you can read the attribute from the top level of a physical CD (CD) or from a track of a physical CD (TR) and from the top level of a physical DVD (DVD) or a title or chapter of a physical DVD (CH).

The fifth column provides a description of the attribute.

**Note** In addition to the attribute names in this table, you can also use the attribute names listed in Attribute Aliases. In many cases, using an alias name allows you to avoid an attribute name that contains a slash character ("/").

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>RW</th>
<th>Media Library</th>
<th>CD/ DVD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>RW</td>
<td>radio</td>
<td>CD/ DVD</td>
<td>Short description of the radio station.</td>
</tr>
<tr>
<td>AcquisitionTime</td>
<td>RO</td>
<td>audio; video;</td>
<td></td>
<td>Date and time the item was added to Media Library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>playlist; radio;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlbumArtistSortOrder</td>
<td>RO</td>
<td>audio</td>
<td></td>
<td>Value for sorting the Album Artist column Media Library.</td>
</tr>
<tr>
<td>AlbumID</td>
<td>RO</td>
<td>audio</td>
<td></td>
<td>Unique identifier for the album (sorted by title when queried by MediaCollection.getAttributeStringCol)</td>
</tr>
<tr>
<td>AlbumIDAlbumArtist</td>
<td>RO</td>
<td>audio</td>
<td></td>
<td>Unique identifier for the album (sorted by artist when queried by MediaCollection.getAttributeStringCol)</td>
</tr>
<tr>
<td>AlbumTitleSortOrder</td>
<td>RO</td>
<td>audio</td>
<td></td>
<td>Value for sorting the Album Title column Media Library.</td>
</tr>
<tr>
<td>Author</td>
<td>RW</td>
<td>audio; video;</td>
<td></td>
<td>Name of a media artist or actor associated the content.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>playlist; radio;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AuthorSortOrder</td>
<td>RO</td>
<td>audio; video</td>
<td></td>
<td>Value for sorting the Author column in Media Library.</td>
</tr>
<tr>
<td>Field</td>
<td>Read/Write</td>
<td>RO</td>
<td>RW</td>
<td>RO</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>AverageLevel</td>
<td>RW</td>
<td>audio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitrate</td>
<td>RO</td>
<td>audio; video; radio; other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BuyNow</td>
<td>RO</td>
<td>audio</td>
<td>CD; TR; DVD; CH</td>
<td></td>
</tr>
<tr>
<td>BuyTickets</td>
<td>RO</td>
<td>audio</td>
<td>DVD; CH</td>
<td></td>
</tr>
<tr>
<td>CallLetters</td>
<td>RO</td>
<td>radio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDTrackEnabled</td>
<td>RW</td>
<td>TR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copyright</td>
<td>RO</td>
<td>audio; video</td>
<td>CD; TR; DVD; CH</td>
<td></td>
</tr>
<tr>
<td>CurrentBitrate</td>
<td>RO</td>
<td>audio; video</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>RW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td>RO</td>
<td>audio; video; other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVDID</td>
<td>RO</td>
<td>video</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileSize</td>
<td>RO</td>
<td>audio; video; other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileType</td>
<td>RO</td>
<td>audio; video; playlist; radio; other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>RO</td>
<td>radio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is_Protected</td>
<td>RO</td>
<td>audio; video</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsVBR</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeadPerformer</td>
<td>RW</td>
<td>DVD; CH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>RO</td>
<td>radio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediaType</td>
<td>RO</td>
<td>audio; video; playlist; radio; other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModifiedBy</td>
<td>RO</td>
<td>TR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoreInfo</td>
<td>RO</td>
<td>audio</td>
<td>CD; TR; DVD; CH</td>
<td></td>
</tr>
<tr>
<td>PeakValue</td>
<td>RW</td>
<td>audio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Media</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ProviderLogoURL</td>
<td>RO</td>
<td>audio</td>
<td>DVD; CH Address of the logo of the provider of the attribute values.</td>
<td></td>
</tr>
<tr>
<td>ProviderURL</td>
<td>RO</td>
<td>audio</td>
<td>DVD; CH Address of the home page of the provider attribute values.</td>
<td></td>
</tr>
<tr>
<td>RadioBand</td>
<td>RO</td>
<td>radio</td>
<td>FM or AM or Net.</td>
<td></td>
</tr>
<tr>
<td>RadioFormat</td>
<td>RO</td>
<td>radio</td>
<td>Description of the type of content provided by the radio station.</td>
<td></td>
</tr>
<tr>
<td>RatingOrg</td>
<td>RO</td>
<td>DVD; CH Name of the provider of the rating value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RecordingTime</td>
<td>RW</td>
<td>audio; video</td>
<td>Date of the original recording, for items with this date different from the release date.</td>
<td></td>
</tr>
<tr>
<td>ReleaseDate</td>
<td>RW</td>
<td>audio; video</td>
<td>CD; TR; DVD; CH Date of the original release of the item.</td>
<td></td>
</tr>
<tr>
<td>RequestState</td>
<td>RW</td>
<td>audio</td>
<td>Media information request state.</td>
<td></td>
</tr>
<tr>
<td>SortAttribute</td>
<td>RW</td>
<td>playlist</td>
<td>Specifies the attribute to use for sorting the playlist; see Playlist.setItemInfo.</td>
<td></td>
</tr>
<tr>
<td>SourceURL</td>
<td>RO</td>
<td>audio; video; playlist; radio; other</td>
<td>Address of the item.</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>RW</td>
<td>audio; video; playlist; radio; other</td>
<td>CD; TR; DVD; CH Title of the content.</td>
<td></td>
</tr>
<tr>
<td>titleNum</td>
<td>RO</td>
<td>CH</td>
<td>Title number of the item; this number is 1-10.</td>
<td></td>
</tr>
<tr>
<td>TitleSortOrder</td>
<td>RO</td>
<td>audio; video; playlist; radio; other</td>
<td>Value for sorting the Title column in Media Library.</td>
<td></td>
</tr>
<tr>
<td>TotalDuration</td>
<td>RO</td>
<td>CD; TR</td>
<td>Playing duration of the item.</td>
<td></td>
</tr>
<tr>
<td>UserCustom1</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Arbitrary text.</td>
<td></td>
</tr>
<tr>
<td>UserCustom2</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Arbitrary text.</td>
<td></td>
</tr>
<tr>
<td>UserEffectiveRating</td>
<td>RO</td>
<td>audio; video; playlist; other</td>
<td>Rating computed by the Player based on how often the item has been played.</td>
<td></td>
</tr>
<tr>
<td>UserLastPlayedTime</td>
<td>RO</td>
<td>audio; video; playlist; other</td>
<td>Date and time the item was most recently played.</td>
<td></td>
</tr>
<tr>
<td>UserPlayCount</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountAfternoon</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played between noon and 17:00 local time.</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountEvening</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played between 17:00 and 22:00 local time.</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Mode</td>
<td>Media Types</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountMorning</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played between 06:00 and noon local time.</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountNight</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played between 22:00 and 06:00 local time.</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountWeekday</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played Monday, Tuesday, Wednesday, Thursday, Friday.</td>
<td></td>
</tr>
<tr>
<td>UserPlaycountWeekend</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Number of times the item has been played Saturday or Sunday.</td>
<td></td>
</tr>
<tr>
<td>UserRating</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Rating specified by the user in the Media Library user interface.</td>
<td></td>
</tr>
<tr>
<td>UserServiceRating</td>
<td>RW</td>
<td>audio; playlist; video</td>
<td>Reserved for future use.</td>
<td></td>
</tr>
<tr>
<td>WM/AlbumArtist</td>
<td>RW</td>
<td>audio CD; TR</td>
<td>Name of the primary artist for the album.</td>
<td></td>
</tr>
<tr>
<td>WM/AlbumCoverURL</td>
<td>RW</td>
<td></td>
<td>Address of a Web page that contains an image of the album cover and information about the album.</td>
<td></td>
</tr>
<tr>
<td>WM/AlbumTitle</td>
<td>RW</td>
<td>audio CD; TR</td>
<td>Title of the album on which the content was originally released.</td>
<td></td>
</tr>
<tr>
<td>WM/AudioFileURL</td>
<td>RO</td>
<td></td>
<td>Address of an official Web page with information about the file.</td>
<td></td>
</tr>
<tr>
<td>WM/AudioSourceURL</td>
<td>RO</td>
<td></td>
<td>Address of an official Web page for the media.</td>
<td></td>
</tr>
<tr>
<td>WM/AuthorURL</td>
<td>RO</td>
<td></td>
<td>Address of the author's Web site.</td>
<td></td>
</tr>
<tr>
<td>WM/BeatsPerMinute</td>
<td>RW</td>
<td></td>
<td>Beats per minute of the content.</td>
<td></td>
</tr>
<tr>
<td>WM/Category</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>Category of the content, such as &quot;Pop&quot; or &quot;Opera&quot;.</td>
<td></td>
</tr>
<tr>
<td>WM/Codec</td>
<td>RO</td>
<td></td>
<td>Name of the codec used to encode the content.</td>
<td></td>
</tr>
<tr>
<td>WM/Composer</td>
<td>RW</td>
<td>audio CD; TR</td>
<td>Name of the composer.</td>
<td></td>
</tr>
<tr>
<td>WM/Conductor</td>
<td>RW</td>
<td>audio TR</td>
<td>Name of the conductor.</td>
<td></td>
</tr>
<tr>
<td>WM/ContentDistributor</td>
<td>RO</td>
<td>audio; video</td>
<td>Name of the distributor of the item. When containing a value for this attribute is added, a new subfolder of the Premium Services is created.</td>
<td></td>
</tr>
<tr>
<td>WM/ContentGroupDescription</td>
<td>RW</td>
<td>audio</td>
<td>Description of the content group, a collection of media items such as a CD boxed set.</td>
<td></td>
</tr>
</tbody>
</table>

The name of the new subfolder is derived from the attribute value. For example, an audio having a value of "Proseware" for WM/ContentDistributor will appear in Media Library in the folder named Premium Services/Music/Proseware.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Read/Write</th>
<th>Media Types</th>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM/Director</td>
<td>RW</td>
<td>video</td>
<td>DVD; CH</td>
<td>Name of the director.</td>
</tr>
<tr>
<td>WM/DRM</td>
<td>RO</td>
<td></td>
<td></td>
<td>Whether the content was protected using rights management (DRM).</td>
</tr>
<tr>
<td>WM/EncodedBy</td>
<td>RO</td>
<td></td>
<td></td>
<td>Name of the person or group who encoded content.</td>
</tr>
<tr>
<td>WM/EncodingSettings</td>
<td>RO</td>
<td></td>
<td></td>
<td>Settings used to encode the content.</td>
</tr>
<tr>
<td>WM/EncodingTime</td>
<td>RO</td>
<td>audio; video; playlist</td>
<td></td>
<td>Time at which the content was encoded.</td>
</tr>
<tr>
<td>WM/Genre</td>
<td>RW</td>
<td>audio; video; playlist; other</td>
<td>CD; TR; DVD; CH</td>
<td>Genre of the content.</td>
</tr>
<tr>
<td>WM/GenreID</td>
<td>RO</td>
<td></td>
<td></td>
<td>Genre identifier compliant with the TCO ID3v2.</td>
</tr>
<tr>
<td>WM/InitialKey</td>
<td>RW</td>
<td>audio</td>
<td></td>
<td>Initial key of the music.</td>
</tr>
<tr>
<td>WM/ISRC</td>
<td>RO</td>
<td></td>
<td></td>
<td>International standard recording code (ISRC).</td>
</tr>
<tr>
<td>WM/Language</td>
<td>RW</td>
<td>audio; video; radio</td>
<td></td>
<td>Language of the item.</td>
</tr>
<tr>
<td>WM/Lyrics</td>
<td>RW</td>
<td>audio</td>
<td>TR</td>
<td>Lyrics of the content.</td>
</tr>
<tr>
<td>WM/Lyrics_Synchronised</td>
<td>RW</td>
<td></td>
<td></td>
<td>Lyrics of the content synchronized to time file.</td>
</tr>
<tr>
<td>WM/MCDI</td>
<td>RO</td>
<td>audio</td>
<td>TR</td>
<td>Music CD identifier of the CD from which file or track came.</td>
</tr>
<tr>
<td>WM/MediaClassPrimaryID</td>
<td>RW</td>
<td>audio; video; playlist; radio; other</td>
<td></td>
<td>GUID specifying one of the primary media classes: music; non-music audio; video; other</td>
</tr>
<tr>
<td>WM/MediaClassSecondaryID</td>
<td>RW</td>
<td>audio; video; playlist; radio; other</td>
<td></td>
<td>GUID specifying the secondary media class.</td>
</tr>
<tr>
<td>WM/Mood</td>
<td>RW</td>
<td></td>
<td></td>
<td>Category name for the mood of the content.</td>
</tr>
<tr>
<td>WM/OriginalAlbumTitle</td>
<td>RW</td>
<td></td>
<td></td>
<td>Name of the album on which the track first appeared.</td>
</tr>
<tr>
<td>WM/OriginalArtist</td>
<td>RW</td>
<td></td>
<td></td>
<td>Name of the artist who originally produced the content.</td>
</tr>
<tr>
<td>WM/OriginalFilename</td>
<td>RW</td>
<td></td>
<td></td>
<td>Name of the file from which the content was made.</td>
</tr>
<tr>
<td>WM/OriginalLyricist</td>
<td>RW</td>
<td></td>
<td></td>
<td>Name of the person who wrote the original lyric.</td>
</tr>
<tr>
<td>WM/OriginalReleaseYear</td>
<td>RW</td>
<td></td>
<td></td>
<td>Year during which the content was originally released.</td>
</tr>
<tr>
<td>WM/ParentalRating</td>
<td>RW</td>
<td>audio; video</td>
<td>DVD; CH</td>
<td>Parental rating of the content.</td>
</tr>
<tr>
<td>Field</td>
<td>Read/Write</td>
<td>Types</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>WM/PartOfSet</td>
<td>RW</td>
<td></td>
<td>Part number and the total number of parts set to which the item belongs.</td>
<td></td>
</tr>
<tr>
<td>WM/Period</td>
<td>RW audio</td>
<td>TR</td>
<td>Period of the item.</td>
<td></td>
</tr>
<tr>
<td>WM/Picture</td>
<td>RO</td>
<td></td>
<td>Graphic related to the content.</td>
<td></td>
</tr>
<tr>
<td>WM/PlaylistDelay</td>
<td>RO</td>
<td></td>
<td>Milliseconds of delay that should precede playback of the item in a playlist.</td>
<td></td>
</tr>
<tr>
<td>WM/Producer</td>
<td>RW video</td>
<td></td>
<td>Name of the producer of the content.</td>
<td></td>
</tr>
<tr>
<td>WM/PromotionURL</td>
<td>RW</td>
<td></td>
<td>Address of a Web site offering a promotion related to the content.</td>
<td></td>
</tr>
<tr>
<td>WM/ProtectionType</td>
<td>RW audio; video</td>
<td></td>
<td>Type of protection used on the content.</td>
<td></td>
</tr>
<tr>
<td>WM/Provider</td>
<td>RO audio; video; radio</td>
<td>CD; TR</td>
<td>Name of the provider of the attribute value.</td>
<td></td>
</tr>
<tr>
<td>WM/ProviderRating</td>
<td>RO audio; video</td>
<td>CD; TR; DVD; CH</td>
<td>Rating of the item as assigned by the attribute provider.</td>
<td></td>
</tr>
<tr>
<td>WM/ProviderStyle</td>
<td>RO audio; video</td>
<td>CD; TR</td>
<td>Style of the item as assigned by the attribute provider.</td>
<td></td>
</tr>
<tr>
<td>WM/Publisher</td>
<td>RW audio; video</td>
<td>CD; TR; DVD; CH</td>
<td>Name of the company that published the content.</td>
<td></td>
</tr>
<tr>
<td>WM/RadioStationName</td>
<td>RO</td>
<td></td>
<td>Name of the radio station associated with content.</td>
<td></td>
</tr>
<tr>
<td>WM/RadioStationOwner</td>
<td>RO</td>
<td></td>
<td>Name of the owner of the radio station associated with the content.</td>
<td></td>
</tr>
<tr>
<td>WM/SubscriptionContentID</td>
<td>RO audio; video</td>
<td></td>
<td>Subscription content identifier.</td>
<td></td>
</tr>
<tr>
<td>WM/SubTitle</td>
<td>RW audio</td>
<td></td>
<td>Subtitle of the content.</td>
<td></td>
</tr>
<tr>
<td>WM/Text</td>
<td>RW</td>
<td></td>
<td>Arbitrary text specified by the user.</td>
<td></td>
</tr>
<tr>
<td>WM/ToolName</td>
<td>RO</td>
<td></td>
<td>Name of the application used to create the content.</td>
<td></td>
</tr>
<tr>
<td>WM/ToolVersion</td>
<td>RO</td>
<td></td>
<td>Version of the application used to create the content.</td>
<td></td>
</tr>
<tr>
<td>WM/TrackNumber</td>
<td>RW audio</td>
<td>TR</td>
<td>Track number of the item; this number is 1.</td>
<td></td>
</tr>
<tr>
<td>WM/UniqueFileIdentifier</td>
<td>RW audio</td>
<td>CD; TR</td>
<td>String that uniquely identifies the item.</td>
<td></td>
</tr>
<tr>
<td>WM/UserWebURL</td>
<td>RO</td>
<td></td>
<td>Address and description of a Web site.</td>
<td></td>
</tr>
<tr>
<td>WM/WMCollectionGroupID</td>
<td>RW audio</td>
<td>CD; TR</td>
<td>GUID identifying the group containing the collection to which the item belongs.</td>
<td></td>
</tr>
<tr>
<td>WM/WMCollectionID</td>
<td>RW audio</td>
<td>CD; TR</td>
<td>GUID identifying the collection to which the item belongs.</td>
<td></td>
</tr>
<tr>
<td>WM/WMCContentID</td>
<td>RW audio</td>
<td>TR</td>
<td>GUID identifying the content.</td>
<td></td>
</tr>
<tr>
<td>WM/Writer</td>
<td>RW audio; video</td>
<td></td>
<td>Name of the writer who wrote the words in the content.</td>
<td></td>
</tr>
</tbody>
</table>
About the WM/MediaClassSecondaryID Attribute

In addition to the attribute values documented in the Windows Media Format SDK, Windows Media Player 9 Series supports two globally unique identifiers (GUIDs) for the WM/MediaClassSecondaryID attribute. The following table lists the GUIDs and their descriptions.

<table>
<thead>
<tr>
<th>GUID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0E20D5C-CAD6-4F66-9FA1-6018830F1DCC</td>
<td>The media object represents a static playlist.</td>
</tr>
<tr>
<td>EB0BAFB6-3C4F-4C31-AA39-95C7B8D7831D</td>
<td>The media object represents an auto playlist.</td>
</tr>
</tbody>
</table>

See Also

- Attribute Aliases
- Media File Attributes
- Media Item Attributes
- Multi-Valued Attributes
- Playlist Attributes

Multi-Valued Attributes

The following list shows the attributes that may have multiple values. To retrieve all of the values for a multi-valued attribute, you must use the Media.getItemInfoByType method, not the Media.getItemInfo method.

- Author
- Description
- WM/Composer (or the alias Composer)
- WM/Conductor (or the alias Conductor)
- WM/Genre (or the alias Genre)
- WM/Language (or the alias Language)
- WM/Lyrics_Synchronised
- WM/UserWebURL
- WM/Writer (or the alias Writer)

See Also

- Attribute Aliases
### Attribute Aliases

The following table lists the attribute names that are actually aliases for a different attribute. You can use either the name in the first column or the name in the second column, but the value is stored as the name in the second column.

In many cases, using an alias name allows you to avoid an attribute name that contains a slash character ("/").

<table>
<thead>
<tr>
<th><strong>Attribute Name</strong></th>
<th><strong>Alias For</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Author</td>
</tr>
<tr>
<td>Album</td>
<td>WM/AlbumTitle</td>
</tr>
<tr>
<td>AlbumArtist</td>
<td>WM/AlbumArtist</td>
</tr>
<tr>
<td>Artist</td>
<td>Author</td>
</tr>
<tr>
<td>Composer</td>
<td>WM/Composer</td>
</tr>
<tr>
<td>Conductor</td>
<td>WM/Conductor</td>
</tr>
<tr>
<td>ContentDistributor</td>
<td>WM/ContentDistributor</td>
</tr>
<tr>
<td>ContentGroupDescription</td>
<td>WM/ContentGroupDescription</td>
</tr>
<tr>
<td>CreationDate</td>
<td>WM/EncodingTime</td>
</tr>
<tr>
<td>DigitallySecure</td>
<td>Is_Protected</td>
</tr>
<tr>
<td>Director</td>
<td>WM/Director</td>
</tr>
<tr>
<td>Genre</td>
<td>WM/Genre</td>
</tr>
<tr>
<td>InitialKey</td>
<td>WM/InitialKey</td>
</tr>
<tr>
<td>Label</td>
<td>WM/Publisher</td>
</tr>
<tr>
<td>Language</td>
<td>WM/Language</td>
</tr>
</tbody>
</table>
Lyrics  WM/Lyrics
MediaClassPrimaryID  WM/MediaClassPrimaryID
MediaClassSecondaryID  WM/MediaClassSecondaryID
MetadataSource  WM/Provider
Mood  WM/Mood
MPAARating  WM/ParentalRating
Name  Title
OriginalIndex  WM/TrackNumber
OriginalIndexLeft  WM/TrackNumber
Period  WM/Period
PlayCount  UserPlayCount
ProducedBy  WM/Producer
ProtectionType  WM/ProtectionType
Rating  WM/ProviderRating
ReleasedBy  WM/Publisher
Size  FileSize
Studio  WM/Publisher
Style  WM/ProviderStyle
SubscriptionContentID  WM/SubscriptionContentID
SubTitle  WM/SubTitle
TOC  WM/MCDI
UniqueFileIdentifier  WM/UniqueFileIdentifier
WMCollectionGroupID  WM/WMCollectionGroupID
WMCollectionID  WM/WMCollectionID
WMContentID  WM/WMContentID
Writer  WM/Writer

See Also

- Available Attributes
- Media File Attributes
- Media Item Attributes
- Multi-Valued Attributes
Media File Attributes

The following table lists the attributes that are stored in the media file, if they are present.

The first column shows the name that you use with methods such as `Media.GetItemInfo` and `Media.GetItemInfoByType`.

The second column shows whether the attribute is stored in both the file and Media Library (ML) or only in the file (F).

The third column shows the constant that you use when programming with the Windows Media Format SDK. You do not require this information to program with the Windows Media Player SDK.

**Note** The file attributes that you can access using the Windows Media Player SDK are a subset of the file attributes that you can access using the Windows Media Format SDK.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Location</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>ML</td>
<td>g_wszWMAuthor</td>
</tr>
<tr>
<td>AverageLevel</td>
<td>ML</td>
<td>g_wszAverageLevel</td>
</tr>
<tr>
<td>Bitrate</td>
<td>ML</td>
<td>g_wszWMCurrentBitrate</td>
</tr>
<tr>
<td>Copyright</td>
<td>ML</td>
<td>g_wszWMCopyright</td>
</tr>
<tr>
<td>Description</td>
<td>F</td>
<td>g_wszWMDescription</td>
</tr>
<tr>
<td>Duration</td>
<td>ML</td>
<td>g_wszWMDuration</td>
</tr>
<tr>
<td>DVDID</td>
<td>ML</td>
<td>g_wszWMDVDID</td>
</tr>
<tr>
<td>FileSize</td>
<td>ML</td>
<td>g_wszWMFileSize</td>
</tr>
<tr>
<td>Is_Protected</td>
<td>ML</td>
<td>g_wszWMProtected</td>
</tr>
<tr>
<td>IsVBR</td>
<td>F</td>
<td>g_wszWMIIsVBR</td>
</tr>
<tr>
<td>ModifiedBy</td>
<td>F</td>
<td>g_wszWMMModifiedBy</td>
</tr>
<tr>
<td>PeakValue</td>
<td>ML</td>
<td>g_wszPeakValue</td>
</tr>
<tr>
<td>Title</td>
<td>ML</td>
<td>g_wszWMTitle</td>
</tr>
<tr>
<td>WM/AlbumArtist</td>
<td>ML</td>
<td>g_wszWMAAlbumArtist</td>
</tr>
<tr>
<td>WM/AlbumCoverURL</td>
<td>F</td>
<td>g_wszWMAAlbumCoverURL</td>
</tr>
<tr>
<td>Metadata Field</td>
<td>Type</td>
<td>Representation</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
<td>-----------------</td>
</tr>
<tr>
<td>WM/AlbumTitle</td>
<td>ML</td>
<td>g_wszWMAlbumTitle</td>
</tr>
<tr>
<td>WM/AudioFileURL</td>
<td>F</td>
<td>g_wszWMAudioFileURL</td>
</tr>
<tr>
<td>WM/AudioSourceURL</td>
<td>F</td>
<td>g_wszWMAudioSourceURL</td>
</tr>
<tr>
<td>WM/AuthorURL</td>
<td>F</td>
<td>g_wszWMAuthorURL</td>
</tr>
<tr>
<td>WM/BeatsPerMinute</td>
<td>F</td>
<td>g_wszWMBeatsPerMinute</td>
</tr>
<tr>
<td>WM/Category</td>
<td>ML</td>
<td>g_wszWMCategory</td>
</tr>
<tr>
<td>WM/Codec</td>
<td>F</td>
<td>g_wszWMCodec</td>
</tr>
<tr>
<td>WM/Composer</td>
<td>ML</td>
<td>g_wszWMComposer</td>
</tr>
<tr>
<td>WM/Conductor</td>
<td>ML</td>
<td>g_wszWMConductor</td>
</tr>
<tr>
<td>WM/ContentDistributor</td>
<td>ML</td>
<td>g_wszWMContentDistributor</td>
</tr>
<tr>
<td>WM/ContentGroupDescription</td>
<td>ML</td>
<td>g_wszWMContentGroupDescription</td>
</tr>
<tr>
<td>WM/Director</td>
<td>ML</td>
<td>g_wszWMDirector</td>
</tr>
<tr>
<td>WM/DRM</td>
<td>F</td>
<td>g_wszWMDRM</td>
</tr>
<tr>
<td>WM/EncodedBy</td>
<td>F</td>
<td>g_wszWMEncodedBy</td>
</tr>
<tr>
<td>WM/EncodingSettings</td>
<td>F</td>
<td>g_wszWMEncodingSettings</td>
</tr>
<tr>
<td>WM/Genre</td>
<td>ML</td>
<td>g_wszWMGenre</td>
</tr>
<tr>
<td>WM/GenreID</td>
<td>F</td>
<td>g_wszWMGenreID</td>
</tr>
<tr>
<td>WM/InitialKey</td>
<td>ML</td>
<td>g_wszWMInitialKey</td>
</tr>
<tr>
<td>WM/ISRC</td>
<td>F</td>
<td>g_wszWMISRC</td>
</tr>
<tr>
<td>WM/Language</td>
<td>ML</td>
<td>g_wszWMLanguage</td>
</tr>
<tr>
<td>WM/Lyrics</td>
<td>ML</td>
<td>g_wszWMLyrics</td>
</tr>
<tr>
<td>WM/Lyrics_Synchronised</td>
<td>ML</td>
<td>g_wszWMLyrics_Synchronised</td>
</tr>
<tr>
<td>WM/MCDI</td>
<td>ML</td>
<td>g_wszWMMCDI</td>
</tr>
<tr>
<td>WM/MediaClassPrimaryID</td>
<td>ML</td>
<td>g_wszWMMediaClassPrimaryID</td>
</tr>
<tr>
<td>WM/MediaClassSecondaryID</td>
<td>ML</td>
<td>g_wszWMMediaClassSecondaryID</td>
</tr>
<tr>
<td>WM/Mood</td>
<td>F</td>
<td>g_wszWMood</td>
</tr>
<tr>
<td>WM/OriginalAlbumTitle</td>
<td>F</td>
<td>g_wszWMOriignalAlbumTitle</td>
</tr>
<tr>
<td>WM/OriginalArtist</td>
<td>F</td>
<td>g_wszWMOriignalArtist</td>
</tr>
<tr>
<td>WM/OriginalFilename</td>
<td>F</td>
<td>g_wszWMOriignalFilename</td>
</tr>
<tr>
<td>WM/OriginalLyricist</td>
<td>F</td>
<td>g_wszWMOriignalLyricist</td>
</tr>
<tr>
<td>WM/OriginalReleaseYear</td>
<td>F</td>
<td>g_wszWMOriignalReleaseYear</td>
</tr>
<tr>
<td>WM/ParentalRating</td>
<td>ML</td>
<td>g_wszWMParentalRating</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
<td>-----------------</td>
</tr>
<tr>
<td>WM/PartOfSet</td>
<td>F</td>
<td>g_wszWMPartOfSet</td>
</tr>
<tr>
<td>WM/Period</td>
<td>ML</td>
<td>g_wszWMPeriod</td>
</tr>
<tr>
<td>WM/Picture</td>
<td>F</td>
<td>g_wszWMPicture</td>
</tr>
<tr>
<td>WM/PlaylistDelay</td>
<td>F</td>
<td>g_wszWMPlaylistDelay</td>
</tr>
<tr>
<td>WM/Producer</td>
<td>ML</td>
<td>g_wszWMProducer</td>
</tr>
<tr>
<td>WM/PromotionURL</td>
<td>F</td>
<td>g_wszWMPromotionURL</td>
</tr>
<tr>
<td>WM/ProtectionType</td>
<td>ML</td>
<td>g_wszWMProtectionType</td>
</tr>
<tr>
<td>WM/Provider</td>
<td>ML</td>
<td>g_wszWMProvider</td>
</tr>
<tr>
<td>WM/ProviderRating</td>
<td>ML</td>
<td>g_wszWMProviderRating</td>
</tr>
<tr>
<td>WM/ProviderStyle</td>
<td>ML</td>
<td>g_wszWMProviderStyle</td>
</tr>
<tr>
<td>WM/Publisher</td>
<td>ML</td>
<td>g_wszWMPublisher</td>
</tr>
<tr>
<td>WM/RadioStationName</td>
<td>F</td>
<td>g_wszWMRadioStationName</td>
</tr>
<tr>
<td>WM/RadioStationOwner</td>
<td>F</td>
<td>g_wszWMRadioStationOwner</td>
</tr>
<tr>
<td>WM/SubTitle</td>
<td>ML</td>
<td>g_wszWMSubTitle</td>
</tr>
<tr>
<td>WM/Text</td>
<td>F</td>
<td>g_wszWMText</td>
</tr>
<tr>
<td>WM/ToolName</td>
<td>F</td>
<td>g_wszWMToolName</td>
</tr>
<tr>
<td>WM/ToolVersion</td>
<td>F</td>
<td>g_wszWMToolVersion</td>
</tr>
<tr>
<td>WM/TrackNumber</td>
<td>ML</td>
<td>g_wszWMTrackNumber</td>
</tr>
<tr>
<td>WM/UniqueFileIdentifier</td>
<td>ML</td>
<td>g_wszWMUniqueFileIdentifier</td>
</tr>
<tr>
<td>WM/UserWebURL</td>
<td>F</td>
<td>g_wszWMUserWebURL</td>
</tr>
<tr>
<td>WM/WMCollectionGroupID</td>
<td>ML</td>
<td>g_wszWMWMCollectionGroupID</td>
</tr>
<tr>
<td>WM/WMCollectionID</td>
<td>ML</td>
<td>g_wszWMWMCollectionID</td>
</tr>
<tr>
<td>WM/WMContentID</td>
<td>ML</td>
<td>g_wszWMWMContentID</td>
</tr>
<tr>
<td>WM/Writer</td>
<td>ML</td>
<td>g_wszWMWriter</td>
</tr>
<tr>
<td>WM/Year</td>
<td>F</td>
<td>g_wszWMYear</td>
</tr>
</tbody>
</table>

See Also

- Attribute Aliases
- Available Attributes
- Media Item Attributes
- Multi-Valued Attributes
Managing Playlists

A Playlist object represents a set of media items. You use properties and methods of the object to manipulate the media items it contains and to retrieve metadata about the playlist. To manage playlists themselves, you use the PlaylistCollection object and the MediaCollection object.

The following topics describe how to manage playlists using the Player control object model:

- Playlist Attributes
- Playlists and Media Items
- Playlists and the PlaylistCollection Object
- Static and Auto Playlists
- Playlists and the MediaCollection Object

See Also

- Managing Media Items
- Playlist Object
- Working with Media Library

Playlist Attributes

Playlists have metadata information called attributes, just as media items have attributes. You can retrieve the names and values of playlist attributes and display them in your user interface, or your code can take actions based on the value of an attribute.

Playlists are defined in files organized in an XML format, and particular elements in the file define playlist attributes. Some attribute elements are well-know; the author of the metafile can also define arbitrary attributes. For more information about attribute elements in playlist files, see Retrieving Metadata.

Media Library may provide additional playlist attributes, such as SourceURL or UserLastPlayedTime. For
more information about the Media Library playlist attributes, see Available Attributes

The Playlist.attributeCount property retrieves the number of attributes associated with the playlist. The Playlist.attributeName property retrieves the name of an attribute based on its index, and the Playlist.getItemInfo method retrieves the value of an attribute based on its name.

You can modify the value of an attribute of the current playlist with the Playlist.setItemInfo method. A special use of the setItemInfo method is to sort the items in the playlist, using the SortAttribute attribute. The following JScript example sorts a playlist by the values of the UserLastPlayedTime attribute. The variable playlist is a reference to a Playlist object.

playlist.setItemInfo("SortAttribute", "UserLastPlayedTime");

The following JScript sample demonstrates retrieving playlist attributes. You can copy this sample to a text file and save the file with the .htm file name extension. When you load the file in Internet Explorer, the top list box displays the playlists in Media Library. When you select one of the playlists, the getSelected function fills the bottom list box with the attributes of the selected playlist.

<HTML>
<HEAD>
<TITLE>Playlist Attributes Sample</TITLE>
</HEAD>

<BODY onLoad="getPlaylists()">
<OBJECT ID="wmpPlayer" NAME="wmpPlayer" HEIGHT="1" WIDTH="1"
CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
<PARAM NAME="uimode" VALUE="invisible">
</OBJECT>

<CENTER>
<form ID="frmTest" NAME="frmTest">
Playlists in Media Library<br><br>
<select ID="lstPlaylists" NAME="lstPlaylists"
SIZE="10" STYLE="width:400px"
onChange="getSelected()">
</select><br><br><br>
Attributes in selected playlist<br><br>
<select ID="lstAttributes" NAME="lstAttributes"
SIZE="10" STYLE="width:400px">
</select>
</form>
</CENTER>

<script LANGUAGE="JScript">
!---
function getPlaylists()
{
  // Get a PlaylistArray object containing all of the
  // playlists in Media Library.
  var selPlaylists = document.frmTest.lstPlaylists;
  var plCollection = wmpPlayer.playlistCollection;
  var plArray = plCollection.getAll();

  // Add the name of each playlist to the SELECT element.
  var playlistCount = plArray.count;
  for (var i = 0; i < playlistCount; i++)
  {
    selPlaylists.options[selPlaylists.options.length]=
    new Option(plArray.item(i).name, i);
  }

  // Add the name of each attribute to the SELECT element.
  var attributeNameCount = plArray.item(0).attributes.length;
  for (var j = 0; j < attributeNameCount; j++)
  {
    var attributeName = plArray.item(0).attributes.item(j).name;
    if (attributeName != "SortAttribute")
    {
      lstAttributes.options[lstAttributes.options.length]=
      new Option(attributeName, attributeName);
    }
  }

  // Get the selected playlist.
  var selectedPlaylist = plCollection.get(plArray.item(0));

  // Get the attribute values.
  var attributeValues = selectedPlaylist.getAttributeValue("UserLastPlayedTime");

  // Display the attribute values.
  for (var k = 0; k < attributeValues.length; k++)
  {
    document.frmTest.doc corpo(n).
  }
}
</script>
</BODY>
function getSelected()
{
    // Get a PlaylistArray object containing the playlists
    // with the specified name.
    var selPlaylists = document.frmTest.lstPlaylists;
    var playlistName = selPlaylists.options(selPlaylists.selectedIndex).text;
    var plCollection = wmpPlayer.playlistCollection;
    var plArray = plCollection.getByName(playlistName);
    // Assume that there is only one playlist in the array.
    var pl = plArray.item(0);

    // For each attribute associated with the playlist,
    // add the name and value to the SELECT element.
    var selAttributes = document.frmTest.lstAttributes;
    selAttributes.options.length = 0;
    var attribCount = pl.attributeCount;
    for (var i = 0; i < attribCount; i++)
    {
        var attribName = pl.attributeName(i);
        var attribValue = pl.getItemInfo(attribName);
        selAttributes.options[selAttributes.options.length]=
            new Option(attribName + " -- " + attribValue, i);
    }
} -->
</SCRIPT>
</BODY>
</HTML>

See Also

- Managing Playlists
- Media Item Attributes
- Playlist Object

Retrieving Media Items

For an existing playlist, you can read the Playlist.count property to determine how many media items are in the playlist, and you can get a reference to the Media object corresponding to a specific item using the Playlist.item...
The following JScript example retrieves an object reference to a specific media item. (Throughout this topic, the variable `playlist` is a reference to a `Playlist` object.)

```javascript
var oMedia = playlist.item(5);
```

The following JScript example retrieves the names of all the media items in a playlist and puts them in a list box on a Web page. The variable `selItems` is a reference to a SELECT element.

```javascript
var itemCount = playlist.count;
for (var i = 0; i < itemCount; i++){
    selItems.options[selItems.options.length] =
        new Option(playlist.item(i).name, i);
}
```

### Adding Items to a Playlist

You can add a media item to the end of a playlist or at a specific position in a playlist, using the `Playlist.appendItem` and `Playlist.insertItem` methods. The following JScript example demonstrates both techniques by adding the current media item twice, at both the end and the beginning of a playlist. The Player control was created with ID="player".

```javascript
var currMedia = player.currentMedia;
playlist.appendItem(currMedia);
playlist.insertItem(0, currMedia);
```

When you create a new, empty playlist by using the `PlaylistCollection.newPlaylist` method, you can add media items to it by repeatedly calling the `Playlist.appendItem` method.

### Manipulating Media Items in a Playlist

You can change the position of a media item in the playlist using the `Playlist.moveItem` method. You specify the item by its current index and you specify the new index. The following JScript example moves an item from index 5 to index 0 within a playlist.

```javascript
playlist.moveItem(5, 0);
```

You can remove a media item from the playlist by using the `Playlist.removeItem` method. Note that if the removed item was not the final item in the playlist, the index values of the subsequent items change. The following JScript example removes the specified item. The variable `oMedia` is a reference to a `Media` object.

```javascript
playlist.removeItem(oMedia);
```

**Note** Users can change the contents of a playlist outside of your application. Whenever you manipulate the items in a playlist, you should monitor and handle the playlist-related events of the Player control to assure that your code works correctly.

### See Also

- Managing Playlists
- Media Object
- Playlist Object
Playlists and the PlaylistCollection Object

The **PlaylistCollection** object gives you access to certain special playlists and has methods for creating new, empty playlists and new playlists from metafiles.

Working with Existing Playlists

The **PlaylistCollection.getAll** and **PlaylistCollection.getByName** methods each return a **PlaylistArray** object, which can contain multiple playlists.

The **PlaylistCollection.getAll** method returns all of the existing playlists that are in **Media Library**. For example, you can call this method and then retrieve the playlists in the **PlaylistArray** object to determine whether a given playlist name has already been used, or to display all of the playlists to the user. The sample code in [Playlist Attributes](#) uses the **getAll** method.

The **PlaylistCollection.getByName** method returns all of the playlists with a given name. You can use this method to handle each of those playlists separately.

You can also use the **getByName** method to retrieve a unique playlist by name. In that case, the **PlaylistArray** object has only one element. The following JScript example demonstrates this technique; the Player control was created with ID="player".

```javascript
var playlist = 
    player.playlistCollection.getByName("PlaylistName").item(0);
```

Working with New Playlists

You can use the **PlaylistCollection.newPlaylist** method to create a new, empty playlist. The method returns a reference to the new **Playlist** object. You can then call the **Playlist.appendItem** method to add media items to the playlist.

You can also create a new playlist based on a playlist metafile. First, pass the name of the playlist and the path to the metafile to the **Player.newPlaylist** method. That method returns a reference to the new **Playlist** object. Then, pass the new **Playlist** object to the **PlaylistCollection.importPlaylist** method.

Notice the difference between the **PlaylistCollection.newPlaylist** method and the **Player.newPlaylist** method. The **PlaylistCollection** method creates a new, empty playlist and adds it to **Media Library**. The **Player** method creates a new, populated **Playlist** object but does not add it to **Media Library**.

The following JScript example demonstrates importing a playlist from a metafile. The player object was created with ID="player" and two input elements were created with ID="txtPlFile" and ID="txtPlName".
var importedPlaylist
function importPlaylist()
{
    var playlistFile = document.frmInput.txtPlFile.value;
    var playlistName = document.frmInput.txtPlName.value;
    var newPlaylist = player.newPlaylist(playlistName, playlistFile);
    importedPlaylist =
        player.playlistCollection.importPlaylist(newPlaylist);
}
// Do some work with the importedPlaylist reference.

See Also

- Managing Playlists
- Player.newPlaylist
- Playlist.appendItem
- PlaylistArray Object
- PlaylistCollection Object
- Playlists and Media Items

© 2000-2003 Microsoft Corporation. All rights reserved.

Static and Auto Playlists

There are two types of playlists: static playlists, which include specific media items, and auto playlists, which search Media Library every time they are opened and may contain different media items at different times.

To import a static playlist from a metafile, first call Player.newPlaylist to create a Playlist object based on the data in the metafile, and then pass that object to PlaylistCollection.importPlaylist to add the playlist to Media Library.

To import an auto playlist from a metafile, use MediaCollection.add. For more information, see Playlists and the MediaCollection Object.

To import a static playlist from an auto playlist metafile, use Player.newPlaylist and PlaylistCollection.importPlaylist as described earlier. The auto playlist will be executed once and a static playlist will be created based on the result of that execution.

Using an auto playlist to query the user's Media Library is not supported for Web pages that users access over the Internet.

The following JScript sample demonstrates importing an auto playlist metafile as a static playlist. To run this sample, create an auto playlist using the Media Library user interface and then include the correct path to the auto playlist metafile in this code.
<HTML>
<HEAD>
<TITLE>Convert Auto Playlist to Static Playlist</TITLE>
</HEAD>

<BODY onLoad="startUp()">

<OBJECT ID="wmpPlayer" NAME="wmpPlayer" HEIGHT="1" WIDTH="1"
CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
<PARAM NAME="autoStart" VALUE="False">
<PARAM NAME="uimode" VALUE="invisible">
</OBJECT>

<FORM ID="frmOuter" NAME="frmOuter">
<CENTER>

<INPUT TYPE="button" ID="btnAddPlaylist" NAME="btnAddPlaylist"
VALUE="Add static playlist from auto playlist file"
onClick="addStaticPlaylist()">

Playlists in Media Library<br>
<br>

<SELECT ID="lstPlaylists" NAME="lstPlaylists" SIZE="25" STYLE="width:400px">
</SELECT></CENTER>
</FORM>

<SCRIPT LANGUAGE="JScript">
!---
function startUp()
{
  var gotFullAccess = wmpPlayer.settings.requestMediaAccessRights("full");

  if(gotFullAccess == true)
  {
    showPlaylists();
  } else {
    alert("This application will not work without full access. \n" +
    "Please click Refresh and start again.");
  }
}

function showPlaylists()
{
  // Get a reference to the playlists list box.
  var playlistsSelect = document.frmOuter.lstPlaylists;
  playlistsSelect.options.length = 0;

  // Get a PlaylistArray with all the playlists and
  // get the count of its playlists.
  var plCollection = wmpPlayer.playlistCollection;
  var plArray = plCollection.getAll();
  var playlistCount = plArray.count;

  // Put the names of the playlists in the list box.
  for (var i = 0; i < playlistCount; i++)
  {
    playlistsSelect.options[playlistsSelect.options.length] =
    new Option(plArray.item(i).name, i);
  }
}

function addStaticPlaylist()
{
  // Create a playlist object from the auto WPL file.
  var newPlaylist = wmpPlayer.newPlaylist("Static Name",
    "\\myServer\\myPath\\myAutoPlaylist.wpl");

  // Check whether there are items in the playlist.
  // If not, create a <BR> element and add it to the list box.
  //
  // Repeat the above code for the check.
if(newPlaylist.count == 0) {
    alert("Sorry, this playlist is empty.");
} else {
    // Import the new Playlist object.
    var addedPlaylist = wmpPlayer.playlistCollection.importPlaylist(newPlaylist);

    // Refresh the playlists list box.
    showPlaylists();
}

See Also

- Managing Playlists

Playlists and the MediaCollection Object

The MediaCollection object gives you access to a variety of special playlists, and includes a method for creating a new playlist from a metafile.

The following methods retrieve special playlists:

- getAll
- getByAlbum
- getByAttribute
- getByAuthor
- getByGenre
- getName

As their names suggest, these methods retrieve playlists containing all of the media items in Media Library that match certain criteria.

Be careful not to confuse the MediaCollection.getName method with the PlaylistCollection.getName method. The MediaCollection method returns a Playlist object containing all of the media items that have the specified name. The PlaylistCollection method returns a PlaylistArray object containing all of the playlists that have the specified name.
You can use the `MediaCollection.add` method to add playlists as well as media items to **Media Library**. To add a playlist, you pass the method the path to the metafile that defines the playlist. The method always returns a reference to a **Media** object; in order to work with the playlist that you added, get a reference to the **Playlist** object that has the same name as the **Media** object.

The following JScript example demonstrates how to add a playlist from a metafile to **Media Library**. The player object was created with ID="player" and an input element was created with ID="txtPlFile".

```javascript
var addedPlaylist;
function addPlaylist()
{
    // Add the playlist as a media item.
    var playlistFile = document.frmInput.txtPlFile.value;
    var addedMedia = player.mediaCollection.add(playlistFile);

    // Get the playlist object with the same name as
    // the added media object.
    var plArray =
        player.playlistCollection.getByName(addedMedia.name);
    // Assume only one playlist in plArray.
    addedPlaylist = plArray.item(0);
}
// Do some work with the addedPlaylist reference.
```

Static playlists include specific media items. Auto playlists search **Media Library** every time they are opened and may contain different media items at different times. You can add both static and auto playlists to **Media Library** using the `MediaCollection.add` method. You can also add static playlists using the `PlaylistCollection.importPlaylist` method.

**See Also**

- Managing Playlists
- `MediaCollection Object`
- `Playlist Object`
- `PlaylistCollection Object`
- Playlists and the PlaylistCollection Object
- Static and Auto Playlists

© 2000-2003 Microsoft Corporation. All rights reserved.
In Microsoft Word, Excel, and PowerPoint®, you embed the control by selecting **Object** from the **Insert** menu, then choosing **Windows Media Player** from the list of available object types. The Player control appears in the document at the current location. You can then select **Format Control (Format Object in Excel)** from the shortcut menu for the control to adjust the layout, text wrapping style, and other format options. In Word and Excel, you must be in design mode to do this.

Once you have positioned and formatted the control, you can configure it using the **Properties** dialog box, which is accessible from the **Control Toolbox** or from the shortcut menu in design mode for Word and Excel. Here you can specify basic Player control properties such as the control name, the URL of a digital media file, and the user interface mode. Setting the **uiMode** property to "none" hides everything in the control except the video or visualization window, allowing you to add your own buttons and write script code using Visual Basic for Applications (VBA) to handle the button clicks and Player control events.

From the basic **Properties** dialog box, you can also access the more sophisticated **Windows Media Player Control Properties** dialog box by double-clicking the "(Custom)" row or by clicking the ellipsis ("...") button after selecting that row. From this dialog box, you can modify all available Player control properties.

**Note** You must be careful not to take actions in Player control event handlers that will result in the control being destroyed. For example, if you embed the Player control on a slide in a PowerPoint presentation, do not call the PowerPoint **Next** method from the Player **openStateChange** event or any other event.

In addition, you should not set the **Player.URL** property from a Player control event handler.

In FrontPage, add the Player control to a Web page by selecting **Web Component** from the **Insert** menu. In the **Insert Web Component** dialog box, select **Advanced Controls** from the **Component type** list, then select **ActiveX Control** from the list of control choices. In the next window of the dialog box, select **Windows Media Player**. If it is not listed, click **Customize** and select the **Windows Media Player** check box in the **Control** list.

After the Player control is embedded, you can position and resize it, and modify its properties by selecting **ActiveX Control Properties** from the shortcut menu for the control. In the HTML view, the property values that you specify appear in the OBJECT element representing the Player control. The object name appears as the ID attribute, and the control properties appear as PARAM tags. The object name gives you access to the Player control object model, which you can program using Microsoft JScript. For more information, see **Embedding the Player Control in a Web Page**.

**See Also**

- **Player Control Guide**

© 2000-2003 Microsoft Corporation. All rights reserved.
This section describes how to use the Windows Media Player ActiveX control in applications created with Microsoft Visual Basic 6.0. It includes a description of the DVD Player sample application, which is written in Visual Basic.

**Getting Started**

To add the Player control to the toolbox, first select **Components** from the **Project** menu. In the Components dialog box, select the check box next to "Windows Media Player". At the bottom of the dialog box, confirm that the selected file is wmp.dll. After closing the dialog box, you can place an instance of the Player control on your form in the usual ways.

You can set many control properties using the Properties window. To set some properties you must use the Windows Media Player Properties dialog box, which you open using the "(Custom)" item in the Properties window.

**Object References**

You use certain Player control properties to get references to particular objects. For example, the **cdromCollection** property returns a reference to a **CdromCollection** object. You must assign such a reference to a variable that you declared as the corresponding interface. In the case of the **cdromCollection** property, for example, you assign its return value to a variable of type **IWMPCdromCollection**.

Read the **Interfaces** topic in the **Object Model Reference for C++** to identify which objects implement multiple interfaces. In those cases, you must declare an object variable as the highest-numbered interface documented in this SDK in order to have access to all of the properties and methods of that object. For example, you should assign the value of the Player control **currentMedia** property to a variable declared as **IWMPMedia3** to assure that you have access to the **getAttributeCountByType** and **getItemInfoByType** methods.

**Note** The **WindowsMediaPlayer** object implements all of the properties and methods of the **IWMPCore**, **IWMPCore2**, **IWMPCore3**, **IWMPPlayer**, **IWMPPlayer2**, **IWMPPlayer3**, and **IWMPPlayer4** interfaces. You do not need to declare separate variables for any of these interfaces. You can access all of their members using the name you assigned to your **WindowsMediaPlayer** instance.

In the Visual Basic Object Browser you will see many interfaces that are intended for private use by the Player control, including some that support skin developers. You should use only the objects, properties, methods, and events that are documented in this SDK.

**Additional Tips**

- The reference documentation shows JScript syntax. In JScript, arguments passed to methods are always enclosed in parentheses. In Visual Basic 6.0, arguments passed to methods that do not return a value must not be enclosed in parentheses.
- Some properties or methods may not appear in the Auto List code-completion feature in the Visual Basic code editor. You can still use those members by typing their names exactly as they appear in this documentation.
- Manage the visual appearance of the control using the **uimode** property. You can do so in two ways. You can use the **Select a mode** drop-down list in the Windows Media Player Properties dialog box, or you can type the correct value in the Properties window.

  In particular, do not use the **visible** property to hide the control; instead, assign the value "invisible" to the **uimode** property.

**Sample Application**
The DVD Player sample application included in this SDK is written in Visual Basic 6.0. The main purpose of the sample is to demonstrate how to access and manipulate DVD video content, but it also demonstrates using the ActiveX control in a Visual Basic-based application.

The following topics describe the sample:

- DVD Player Start-Up State
- DVD Player Disc Navigation
- Additional Functionality in the DVD Player
- WMPDVD Protocol in the DVD Player

See Also

- Player Control Guide

© 2000-2003 Microsoft Corporation. All rights reserved.

DVD Player Start-Up State

At design time, the ActiveX control is given the name wmpPlayer, but otherwise left in its default state. This means that the autoStart property is True and the URL property has no value.

There are three module-level variables:

Private oSelectedDrive As IWMPCdrom
Private oTopMenu As IWMPPplaylist
Private strProtocol As String

The oSelectedDrive variable will always refer to the drive that is currently selected in the cboDrives combo box. Note that the data type is IWMPCdrom. DVD drives can play both CD and DVD discs; a Cdrom object can therefore represent a CD drive or a DVD drive.

The oTopMenu variable will refer to a Playlist object representing the top-most menu of a DVD disc, the menu containing all of the titles on the disc.

The strProtocol variable will be used with the WMPDVD protocol functionality, described in a later topic.

The following topics describe the start-up code in the application:

- Handling the Form_Load Event
- Enumerating the Drives
- Handling Drive Selection
Handling the Form_Load Event

The Form_Load event procedure takes three actions. First, it determines whether a DVD decoder is available. Windows Media Player does not include a DVD decoder, so a third-party decoder must be installed separately. The procedure calls the available method of the DVD object, which it references through the dvd property of the control.

Second, it calls DisplayDrives, a user-defined procedure that determines which drives on the system can play CDs, and then adds their drive letters to the combo box.

Third, it sets the ListIndex property of the combo box, which raises the cboDrives_Click event.

This is the Form_Load event procedure:

```vba
Private Sub Form_Load()
    If Not wmpPlayer.dvd.isAvailable("dvdDecoder") Then
        MsgBox "No DVD encoder installed.", vbCritical, "WARNING!"
    End If
    DisplayDrives
    cboDrives.ListIndex = 0
End Sub
```

See Also

- DVD Player Start-Up State
Enumerating the Drives

The DisplayDrives procedure discovers the CD drives on the system and adds their drive letters to the cboDrives combo box.

First, it determines how many CD-capable drives are on the system through the count property of the CdromCollection object, which it references through the cdromCollection property of the control.

It uses the count to loop through the collection of drives. For each drive, it retrieves the driveSpecifier property and adds that value to the cboDrives combo box.

This is the DisplayDrives procedure:

```vbscript
Private Sub DisplayDrives()
    Dim iNumDrives As Integer
    Dim i As Integer
    Dim oThisDrive As IWMPCdrom

    iNumDrives = wmpPlayer.cdromCollection.Count
    If Not iNumDrives > 0 Then
        MsgBox "No DVD or CD drive available.", vbCritical, "WARNING!"
    End If

    For i = 0 To iNumDrives - 1
        Set oThisDrive = wmpPlayer.cdromCollection.Item(i)
        cboDrives.AddItem oThisDrive.driveSpecifier
    Next i
End Sub
```

See Also

- [DVD Player Start-Up State](#)
that displays the titles which are on the disc in that drive. The cboDrives_Click event procedure handles this task. The procedure executes for the first time when the combo box **ListIndex** property is set in the Form_Load event procedure.

First, this event procedure stores the new drive selection in the module-level variable **oSelectedDrive**. It references the **CdromCollection** object through the **cdromCollection** property of the control. It passes the combo box **ListIndex** property to the **Item** property of the **CdromCollection** object in order to specify a particular CD-capable drive.

Second, it calls the user-defined procedure **ShowProtocol**, which is related to the WMPDVD protocol functionality and will be discussed in a later topic.

Third, recall that the data type of **oSelectedDrive** is **IWMPCdrom**. The value of that object's **playlist** property is a **Playlist** object containing the items in the top-most menu of the disc in the drive. For a DVD disc, that menu contains the titles on the disc. The event procedure assigns the playlist to the module-level **oTopMenu** variable and also to the **currentPlaylist** property of the Player control. Because the **autoStart** property of the control is True, specifying the current playlist has the effect of beginning playback.

Finally, it calls the user-defined procedure **DisplayPlaylist**, which adds the items from the current playlist to the lstMenuItems list box.

This is the cboDrives_Click procedure:

```vba
Private cboDrives_Click()
    Set oSelectedDrive = wmpPlayer.cdromCollection.Item(cboDrives.ListIndex)
    ShowProtocol
    Set oTopMenu = oSelectedDrive.Playlist
    wmpPlayer.currentPlaylist = oTopMenu
    DisplayPlaylist
End Sub
```

See Also

- [DVD Player Start-Up State](#)

Displaying the Playlist

The user-defined procedure **DisplayPlaylist** simply adds each item in the current playlist to the lstMenuItems list box. The procedure is called each time the current playlist changes.

It is called the first time from the cboDrives_Click event procedure. It is also called from the
wmpPlayer_CdromMediaChange and wmpPlayer_OpenPlaylistSwitch event procedures. The CdromMediaChange event is raised when the user inserts or removes a disc. The OpenPlaylistSwitch event is raised when the current playlist changes for other reasons, for example, when playback moves from an introductory title to the main title.

The DisplayPlaylist procedure starts by assigning the current playlist to the oSelectedPlaylist variable. Depending on the context, oSelectedPlaylist will represent the titles on the disc or the chapters in a particular title.

It then retrieves the Count property and uses that number to enumerate the items in the list. For each item, it retrieves the Name property and adds that value to the list box.

This is the DisplayPlaylist procedure:

Private Sub DisplayPlaylist()
    Dim oSelectedPlaylist As IWMPPlaylist
    Dim iNumItems As Integer
    Dim i As Integer
    Set oSelectedPlaylist = wmpPlayer.currentPlaylist
    iNumItems = oSelectedPlaylist.Count
    If Not iNumItems > 0 Then
        MsgBox "No items in the playlist or" & vbCrLf & "no disc in the drive.", vbCritical, "WARNING!
    End If
    lstMenuItems.Clear
    For i = 0 To iNumItems - 1
        lstMenuItems.AddItem oSelectedPlaylist.Item(i).Name
    Next i
End Sub

See Also

- DVD Player Start-Up State

Summary

This completes the description of the code that executes as soon as you run the DVD Player project. The Form_Load event procedure calls a user-defined procedure to add all of the CD-capable drives to the cboDrives combo box, and then raises the cboDrives_Click event procedure. The cboDrives_Click event procedure sets the top-most menu of the DVD disc as the current playlist, and then calls a user-defined procedure to add the items from the current playlist to the lstMenuItems list box.
The final effect is that the first chapter of the first title on the DVD disc begins playing. If the DVD disc has been previously played on this system, then the disc begins playing at the point where it last stopped playing.

See Also

- DVD Player Start-Up State

DVD Player Disc Navigation

The content of a DVD disc is organized into one or more titles, each of which has one or more chapters. At any given moment during playback, the content comes from a specific chapter.

There is no required standard for the arrangement of titles and chapters. It is typical for a disc to begin with one or two titles of one chapter each, containing copyright and other introductory material. This may be followed by a title with many chapters, containing the primary content of the disc. And the structure may conclude with additional titles containing extra content, such as interviews, short features, and so on.

Because of the lack of requirements, the Windows Media Player object model methods for navigating a disc may have different effects with different DVD discs.

Four of the command buttons in the DVD Player application user interface relate to navigating the content structure of the DVD disc. Two of them display the basic menus of the DVD content structure. One of them navigates back one level in the menu structure. One of them resumes playback from a menu page.

The following topics describes the disc navigation functionality:

- Basic Menus
- Additional Navigation

See Also

- Using Windows Media Player with Visual Basic

© 2000-2003 Microsoft Corporation. All rights reserved.
Basic Menus

In the Windows Media Player object model, the top-most menu is called the top menu. Its items are typically titles. Other menus are called title menus because they typically have the chapter items from one title.

The cmdShowTopMenu command button, which displays "List titles on disc" in the user interface, calls the topMenu method of the DVD object. It references that object through the dvd property of the control. This method will usually navigate to the top-most menu of the disc.

The event procedure first retrieves the availability of the method using the isAvailable property of the DVD object; however, depending on how a particular disc was authored, this method may return True even when the topMenu method has no effect.

This is the cmdShowTopMenu_Click procedure:

Private Sub cmdShowTopMenu_Click()
    If wmpPlayer.dvd.isAvailable("topMenu") Then
        wmpPlayer.dvd.topMenu
    Else
        MsgBox "The topMenu method is not available" & vbCrLf & "at this point on this disc.", vbInformation
    End If
End Sub

The cmdShowTitleMenu command button, which displays "List chapters in title" in the user interface, calls the titleMenu method of the DVD object. It references that object through the dvd property of the control. This method will usually navigate to the chapter menu for the current title.

The event procedure first retrieves the availability of the method using the isAvailable property of the DVD object; however, depending on how a particular disc was authored, this method may return True even when the titleMenu method has no effect.

This is the cmdShowTitleMenu_Click procedure:

Private Sub cmdShowTitleMenu_Click()
    If wmpPlayer.dvd.isAvailable("titleMenu") Then
        wmpPlayer.dvd.titleMenu
    Else
        MsgBox "The titleMenu method is not available" & vbCrLf & "at this point on this disc.", vbInformation
    End If
End Sub

See Also

- DVD Player Disc Navigation
The DVD Player project uses two other **DVD** methods that navigate the menu structure.

The cmdMenuBack command button, which displays "<< Previous menu" in the user interface, calls the **back** method of the **DVD** object. It references that object through the **dvd** property of the control. This method will usually navigate back to the previous menu.

In the event procedure, we first retrieve the availability of the method using the **isAvailable** property of the **DVD** object; however, depending on how a particular disc was authored, this method may return True even when the **back** method has no effect.

This is the `cmdMenuBack_Click` procedure:

```vbscript
Private Sub cmdMenuBack_Click()
    If wmpPlayer.dvd.isAvailable("back") Then
        wmpPlayer.dvd.back
        DisplayPlaylist
    Else
        MsgBox "The Back method is not available" & vbCrLf & _
        "at this point on this disc.", vbInformation
    End If
End Sub
```

The cmdMenuResume command button, which displays "Resume from menu" in the user interface, calls the **resume** method of the **DVD** object. It references that object through the **dvd** property of the control. This method will usually resume playback at the point from which the menu was invoked.

The event procedure first retrieves the availability of the method using the **isAvailable** property of the **DVD** object; however, depending on how a particular disc was authored, this method may return True even when the **resume** method has no effect.

This is the `cmdMenuResume_Click` procedure:

```vbscript
Private Sub cmdMenuResume_Click()
    If wmpPlayer.dvd.isAvailable("resume") Then
        wmpPlayer.dvd.resume
    Else
        MsgBox "The Resume method is not available" & vbCrLf & _
        "at this point on this disc.", vbInformation
    End If
End Sub
```

**See Also**
Additional Functionality in the DVD Player

The DVD Player sample project demonstrates features of the DVD object in addition to the content navigation features.

The application will automatically play a DVD disc if one is already in a drive when the application starts, or if one is inserted while the application is running, because of the code discussed in the topic DVD Player Start-Up State. The user can also use the title or chapter items displayed in the lstMenuItems list box to play a specific portion of the disc.

The user can select an item in the list box and then click the cmdPlaySelected command button, which displays "Play selected item" in the user interface.

The cmdPlaySelected_Click event procedure starts by checking to see whether a list box item is actually selected. This avoids errors when passing an index from the list box to an object model method. If an item is selected, the event procedure uses the ListIndex property of the list box to get a specific media item from the current playlist. It assigns that item to the currentItem property of the control, which it references through the Controls property. Because the autoStart property of the control is True, specifying the current item has the effect of beginning playback with the current item.

Private Sub cmdPlaySelected_Click()
    If lstMenuItems.ListIndex = -1 Then
        MsgBox "Please select an item and then try again.", vbExclamation
        Exit Sub
    End If
    wmpPlayer.Controls.currentItem = _
    wmpPlayer.currentPlaylist.Item(lstMenuItems.ListIndex)
    wmpPlayer.Controls.play
End Sub

The user can also play back a specific item by double-clicking it in the list box. The application implements this functionality by calling the cmdPlaySelected_Click procedure from the lstMenuItems_DblClick event procedure.

Finally, the application displays the current DVD domain whenever it changes. This is the wmpPlayer_DomainChange procedure:

Private Sub wmpPlayer_DomainChange(ByVal strDomain As String)
    lblDomain.Caption = "Domain: " & strDomain
End Sub
WMPDVD Protocol in the DVD Player

You can write an application to play back any DVD disc inserted in a DVD drive using the **Cdrom** and **Playlist** objects as described in previous topics. You can also write an application that plays back DVD media from a local hard disk or network share.

From a file system point of view, a DVD disc has two top-level folders, named AUDIO_TS and VIDEO_TS. The VIDEO_TS folder contains a file named VIDEO_TS.IFO. Video content in DVD format preserves that structure when it is stored on a hard disk.

You can play back DVD media by assigning a value to the Player control **URL** property that specifies the path to a VIDEO_TS.IFO file. The protocol that you use for the URL is WMPDVD.

The text boxes and command button at the bottom of the DVD Player user interface demonstrate using WMPDVD. They use the protocol to navigate a DVD disc in a drive, but can be modified to point to DVD media that is not on a disc.

The following topics describe the WMPDVD functionality in the DVD Player application:

- Creating the Protocol String
- Using the Protocol

See Also

- Using Windows Media Player with Visual Basic
- WMPDVD Protocol
Creating the Protocol String

Whenever the user selects a drive in the cboDrives combo box, the cboDrives_Click event procedure calls the user-defined procedure ShowProtocol. The txtTitle_Change and txtChapter_Change event procedures also call ShowProtocol. It is called for the first time when the Form_Load procedure calls the cboDrives_Click procedure.

The ShowProtocol procedure constructs a valid WMPDVD URL from the current drive letter, the current value of the txtTitle text box, and the current value of the txtChapter text box. It assigns that WMPDVD string to the module-level variable `strProtocol` and then displays it in the caption of the frame.

This is the ShowProtocol procedure:

```vba
Private Sub ShowProtocol()
    strProtocol = "wmpdvd://" & Left$(cboDrives.Text, 1)
    If Trim(txtTitle.Text) <> "" Then
        strProtocol = strProtocol & "/" & Trim(txtTitle.Text)
        If Trim(txtChapter.Text) <> "" Then
            strProtocol = strProtocol & "/" & Trim(txtChapter.Text)
        End If
    End If
End Sub
```

See Also

- WMPDVD Protocol in the DVD Player

Using the Protocol

The user can click the cmdPlayWithProtocol command button, which displays "Play using WMPDVD protocol" in the user interface, after selecting a drive, after entering a title number, or after entering both a title number and a chapter number.
The cmdPlayWithProtocol_Click event procedure first turns on error handling and then assigns the module-level variable `strProtocol` to the Player control `URL` property. The ShowProtocol procedure, described in the previous topic, assures that `strProtocol` is always a valid WMPDVD URL. Because the autoStart property of the control is True, specifying the URL has the effect of beginning playback.

This is the cmdPlayWithProtocol_Click event procedure:

```vbscript
Private Sub cmdPlayWithProtocol_Click()
    wmpPlayer.Error.clearErrorQueue
    On Error GoTo ErrHandler
    wmpPlayer.URL = strProtocol
    Exit Sub

ErrHandler:
    Dim i As Integer
    For i = 0 To wmpPlayer.Error.errorCount - 1
        MsgBox wmpPlayer.Error.Item(i).errorDescription
    Next
End Sub
```

The WMPDVD protocol functionality allows you to build an application that plays back DVD media created on the local computer.

See Also

- WMPDVD Protocol in the DVD Player

© 2000-2003 Microsoft Corporation. All rights reserved.

---

### Embedding the Player Control in a C++ Program

There are several different ways to embed the Windows Media Player control in a custom C++ program. The Microsoft Foundation Classes (MFC) library provides a user-friendly embedding experience similar to Microsoft Visual Basic. This approach does not require extensive experience with the Microsoft Component Object Model (COM), and is suitable for novice C++ programmers. Those who are familiar with COM can implement interfaces provided to run the embedded Player control in remote mode and to customize the user interface by applying a skin definition file.

This information is described in the following topics.
Embedding the Player Control in an MFC Project

This section describes how to embed the Windows Media Player control in a window-based program that you create using Microsoft Visual C++ version 6.0 and the Microsoft Foundation Classes (MFC) Library.

MFC makes it almost as easy to embed ActiveX controls in C++ programs as it is in Visual Basic-based programs. When you install Windows Media Player, the ActiveX control is registered and made available to Microsoft Visual Studio, where you can add it to the Controls palette in the dialog resource editor. From here, you can drag it onto a dialog resource, modify its properties, and add code to manipulate it.

The Player control uses Microsoft Component Object Model (COM) technology to expose a number of interfaces. MFC generates a C++ wrapper class for most of these interfaces, letting you manipulate the control using standard object-oriented programming (OOP) syntax.

The following sections describe the basic steps you use to embed the control, add buttons to provide a custom user interface, and provide code that plays, pauses, and stops your digital media when a button is clicked:

- Adding the Control to Your Project
- Adding the Control to a Dialog Resource
- Adding Code to Manipulate the Control
- Handling Player Control Events
- Using Player Interfaces Without a Wrapper Class
- Using Player Objects With a Wrapper Class
Adding the Control to Your Project

First, start a new dialog-based project using MFC AppWizard (exe), making sure the ActiveX Controls support option is selected.

Next, add the control through the Project menu by selecting Add To Project, then Components and Controls. This displays the Components and Controls Gallery dialog box and places you in the Gallery folder. From here, you can access the Registered ActiveX Controls folder, select Windows Media Player, and insert it into the project.

When you insert the control, the wizard generates a C++ wrapper class for most of the COM interfaces that the control exposes. The primary class is CWMPPlayer4, which corresponds to the IWMPPlayer4 interface. MFC does not automatically generate a wrapper class for certain interfaces. To determine whether a particular interface has a wrapper class, you can browse the ClassView tab in Visual Studio and look for a class name that contains the corresponding Windows Media Player object name. For instance, the DVD object has a wrapper class named "CWMPDVD". Whether or not a particular interface generates a wrapper class in MFC is also documented on the main page for each Player interface in Object Model Reference for C++.

See Also

- Embedding the Player Control in a C++ Program
- Using Player Interfaces Without a Wrapper Class
- Using Player Objects With a Wrapper Class

© 2000-2003 Microsoft Corporation. All rights reserved.
Adding the Control to a Dialog Resource

After you add the control to your project, it appears on the Controls palette. To insert it into a dialog resource, delete the TODO label in the center of the dialog box, then drag the Player control to the form and resize the control border until the entire user interface is visible.

Next, display the Windows Media Player Properties dialog box using the shortcut menu. On the General tab, change the ID to IDC_PLAYER. On the Control tab, type the file name or URL of a digital media file, or use the Browse button to locate a file. Then, because you will provide a custom user interface, select "none" from the Select a mode list. In this mode, only the video or visualization window is displayed.

Next, create three button controls. On the property dialog box for each control, modify its ID and caption. For the button IDs, use IDC_PLAY_BUTTON, IDC_PAUSE_BUTTON, and IDC_STOP_BUTTON. For the captions, use Play, Pause, and Stop.

See Also

- Embedding the Player Control in an MFC Project

Adding Code to Manipulate the Control

Before your code can reference the Player control, you must provide a member variable for it. To do this, open the ClassWizard from the shortcut menu for the control. Next, select the Member Variables tab, and select IDC_PLAYER from the Control IDs list. Finally, click Add Variable, and type m_ctrlPlayer for the member variable name. This member variable is in the Control category and represents an instance of the CWMPPlayer4 class, through which you can access the entire object model.

To add event handlers for the buttons you've created, double-click each one in turn, and accept the default member function name in the Add Member Function dialog box. This generates a function block into which you can type the event handler code shown below. Your functions should be similar to following three, differing only in the dialog class name, which is based on the project name.

```cpp
void CWMPEmbedDlg::OnPlayButton()
{
    m_ctrlPlayer.GetControls().play();
}

void CWMPEmbedDlg::OnPauseButton()
{
    m_ctrlPlayer.GetControls().pause();
}
```
void CWMPEmbedDlg::OnStopButton()
{
    m_ctrlPlayer.GetControls().stop();
}

See Also

- Embedding the Player Control in an MFC Project

Handling Player Control Events

In addition to handling button click events, you can handle any of the events documented in the Player control object model. These events are listed when you select Events from the shortcut menu for the control. To create an event handler for a particular event, select the event from the New Windows messages/events list and click Add and Edit.

The following code example illustrates an event handler for the PlayStateChange event.

void CWMPEmbedDlg::OnPlayStateChangePlayer(long NewState)
{
    if (NewState == 8) {  // playback has completed
        MessageBox("The End");
    }
}

See Also

- Embedding the Player Control in an MFC Project

Using Player Interfaces Without a Wrapper Class
To use Windows Media Player interface for which MFC does not generate a wrapper class, you must first obtain a pointer to parent interface by using the appropriate `IWMPPlayer4` wrapper class accessor method, then use that pointer to `QueryInterface` for the subsequent numbered interface you want to work with. Once you have the pointer to the specific interface you want to use, you can then call its methods through the pointer. The following code example demonstrates using the `get_defaultAudioLanguage` method of the `IWMPSettings2` interface:

```cpp
IWMPPlayer4 m_ctrlPlayer;
IDispatch *pSettings = NULL;
IWMPSettings2 *pSettings2 = NULL;
HRESULT hr = S_OK;

// Get a pointer to the IDispatch interface from CWMPSettings.
pSettings = m_ctrlPlayer.GetSettings().m_lpDispatch;

if( pSettings != NULL )
{
    // Get a pointer to the IWMPSettings2 interface.
    hr = pSettings->QueryInterface(__uuidof( IWMPSettings2 ), reinterpret_cast<void**>( &pSettings2 ));

    if( SUCCEEDED( hr ) )
    {
        long lLangID;
        hr = pSettings2->get_defaultAudioLanguage(&lLangID);
    }

    if( SUCCEEDED( hr ) )
    {
        // Use the lLangID value somehow...
    }

    // Don't forget to call release through the pointers.
    if( pSettings )
    {
        pSettings->Release();
    }

    if( pSettings2 )
    {
        pSettings2->Release();
    }
}
```

For the preceding example to compile, you need to add the include statement for the CWMPSettings class header. The following code would be added to your project's main header file:

```cpp
#include "wmpsettings.h"
```

See Also

- [Embedding the Player Control in an MFC Project](#)
- [Using Player Objects With a Wrapper Class](#)
Using Player Objects With a Wrapper Class

When you browse the generated classes from the ClassView tab of the Workspace pane in Visual Studio, you can see a list of the methods each class supports, including all the Player object model methods and accessor methods for each Player property. These accessor methods have names that begin with "Get" or "Set" followed by the name of the property. For example, to use the Player.controls property of the Player object model to retrieve the Controls object and then call the Controls.play method, you would use the following syntax:

```
m_ctrlPlayer.GetControls().play();
```

Because you are now referencing a wrapper class, you must add an include statement for the class header to your project's main header file. For the CWMPPControls class, you would use the following syntax:

```
#include "wmpcontrols.h"
```

See Also

- Embedding the Player Control in an MFC Project
- Using Player Interfaces Without a Wrapper Class

Remoting the Windows Media Player Control

When you embed the Windows Media Player control in a C++ program, you can use it as a remote extension of the full mode of the Player. This is called "remoting" the Player control, and it lets you provide all the features of the full Player without implementing them yourself. When you remote the control, it shares the same playback engine as the full mode of the Player and your users can switch back and forth between embedded mode (the "docked" state) and full mode (the "undocked" state) while digital media playback continues uninterrupted.

Enabling Remote Embedding

To enable remote embedding of the Player control, your program must implement the IServiceProvider and IWMPRemoteMediaServices interfaces. IServiceProvider is a standard Component Object Model (COM)
interface with a single method called QueryService. Windows Media Player calls this method to retrieve a pointer to an IWMPRemoteMediaServices interface.

IWMPRemoteMediaServices has several methods, but only two of them are directly relevant to remoting. In GetApplicationName, you return the name of your program, which Windows Media Player adds to the Switch to Other Program list on the View menu. In GetServiceType, you indicate the embedding mode of the control by returning a value of either "Remote" or "Local". If a remote connection is successfully established, the get_isRemote method of the IWMPPlayer4 interface returns true.

Docking and Undocking

The IWMPPlayer4 interface also provides access to the IWMPPlayerApplication interface through the get_playerApplication method. Use IWMPPlayerApplication to switch back and forth between the docked and undocked states and to determine the current docked state and the location of the video or visualization display.

The IWMPPlayerApplication::switchToPlayerApplication method undocks the control by opening the full mode of the Player and transferring the video or visualization display to the Now Playing pane. The IWMPPlayerApplication::switchToControl method docks the control by transferring the video or visualization display to your program and closing the full mode of the Player, if it is open. The control can also be docked by selecting a program from the Switch to Other Program list mentioned earlier, or by closing the full mode of the Player. In both cases, any digital media that is playing continues uninterrupted.

Transferring the Video or Visualization Display

A video or visualization can display in only one location at a time. When several programs with embedded, remoted Player controls run simultaneously, they all share the same playback engine and the same instance of the full mode of the Player in the undocked state. In the docked state, however, only one of them can display the video or visualization. In the undocked state, only the full mode of the Player displays the video or visualization. The switchToControl method works in both the docked and undocked states, transferring the video or visualization display to whichever program calls it.

The only difference between the docked and undocked states is the presence of the full mode of the Player and its ownership of the video or visualization display. In the undocked state, all embedded, remoted controls currently running are still visible and their user interfaces are still fully functional. If a video or visualization window is present, however, it is empty. In the docked state, only one of the embedded, remoted controls owns the display, but all user interfaces continue to function.

Hiding or Changing the Control in the Undocked State

You must provide your own implementation if you want to hide or alter the user interface of an embedded control in the undocked state or when your program does not own the display. You can make these alterations when you dock and undock the control or you can make them in response to Windows Media Player events. Because the Player can be docked through the Switch to Other Program menu option, however, it is usually better to provide this functionality in response to events.

You can implement event handlers for the SwitchedToPlayerApplication and SwitchedToControl events, or you can implement a single event handler for the PlayerDockedStateChanged event. In the latter case, you can determine the docked state by calling IWMPPlayerApplication::get_playerDocked. In both cases, use IWMPPlayerApplication::get_hasDisplay to determine whether your program owns the video or visualization display.

Re-Establishing a Remote Connection
In certain circumstances, the connection between a remoted, embedded control and the standalone Player will fail, invalidating your pointers to the Windows Media Player interfaces. The Player will automatically attempt to reconnect, and will fire the **PlayerReconnect** event to signal this attempt. Although the reconnection is automatic, you must provide an event handler for this event if you want to release your invalid pointers and retrieve new ones so that you can access the standalone Player through the new connection.

**Controlling the Undocked Player**

Remote embedding of the Player control lets you extend your program to include everything that is already available in the full mode of Windows Media Player. Although enabling this feature provides a great benefit to your users, you may be wondering whether the undocked, full mode of the Player will ultimately distract your users from your own program.

To retain continuity between your program and the undocked Player, use the **IWMPPPlayerServices** interface. This interface lets you specify both the full mode pane that the Player undocks into and the contents of that pane. Use the `setTaskPane` method to specify the pane, and the `setTaskPaneURL` method to specify a Web page that appears in that pane. If you undock into the **Now Playing** pane, you can specify a user interface plug-in to display there by calling `activateUIPlugin`. This method also activates background and separate window UI plug-ins.

By specifying Web pages for the various panes of the full Player, you can control many aspects of the user experience in the undocked state. You can use these Web pages to display your own branding and advertising, for example, or to replace windowsmedia.com in the **Media Guide** pane with your own media guide.

You can also provide supplementary user interfaces by embedding the Player control in your HTML and writing Microsoft JScript code to manipulate it. The control embedded in the HTML recognizes that the full Player is hosting it and establishes a remote connection automatically. This allows you to manipulate the Player in JScript code as described for metafile Web pages in [Displaying Web Pages in Windows Media Player](#).

All remoted instances of the Player control can manipulate the full mode of the Player regardless of the docked state. Features that have no relevance to the full mode of the Player, however, are ignored until the Player control is docked. This includes properties of the **IWMPPPlayer** and derived interfaces, such as **enabled**, **enableContextMenu**, **uiMode**, and **windowlessVideo**.

**Error Dialog Boxes**

From a remoted Windows Media Player control instance, the **Settings**.**enableErrorDialogs** property behaves in a specific manner. The following rules apply:

- When the Player is undocked (the Windows Media Player application is visible), the **enableErrorDialogs** property is ignored and error dialog boxes are handled by Windows Media Player.
- When the Player is docked, the value specified by each remoted instance of the control for **enableErrorDialogs** applies only to the individual control instance. That is, if a particular control instance specifies a value of "true" for **enableErrorDialogs**, only that instance will display a dialog box when an error occurs if all other instances of the control have specified a value of "false".

**See Also**

- [Embedding the Player Control in a C++ Program](#)
Using Skins with the Windows Media Player Control

When you embed the Player control in a C++ program, you have the option of customizing its user interface by applying a skin definition file to it. A skin definition file is an XML-based document specifying the layout of standard and customizable user interface components and any accompanying graphics. Using Microsoft JScript, you can specify the behavior of these components and manipulate the Player control without the overhead of C++ and COM syntax.

Skins provide an easy way to keep your user interface code and your main program code separate so that they can be maintained and developed independently. You can also reuse skins originally designed for use by the standalone Player in skin mode. Skin code that you design specifically for C++ programs can interact with your programs through a scriptable object that your program can provide.

To enable skin mode for the Player control, your program must implement the IWMPRemoteMediaServices interface. Although using skins with the control and remoting the control go well together, you can use this interface to enable either feature without enabling the other. To disable remoting, simply pass a value of "Local" as the out parameter of the GetServiceType method, and return an HRESULT of E_NOTIMPL from the GetApplicationName method.

To switch the Player control to skin mode, call the IWMPPlayer::put_uiMode method, passing in a value of "custom". Specify the path and file name of the skin definition file to use by returning it from the IWMPRemoteMediaServices::GetCustomUIMode method. If you want to provide a scriptable object for communication between your skin and your program, pass a name and a pointer to an IDispatch pointer as the two out parameters of the IWMPRemoteMediaServices::GetScriptableObject method. Your skin can then make calls to the scriptable object by using the name specified as though it were a global attribute similar to the player global attribute.

A skin applied to a remoted Player control can access the PlayerApplication object using another global attribute called playerApplication. Because the Player.playerApplication property is inaccessible to skins, you must use this global attribute when you want your skin code to manage docking and undocking.

See Also

- Embedding the Player Control in a C++ Program
Embedding the Player Control in a .NET Framework Solution

The Windows Media Player 9 Series ActiveX control supports embedding in .NET Framework applications. The following sections provide information specific to using the Player control with the .NET Framework.

Section | Description
--- | ---
About Primary Interop Assemblies | Provides information about primary interop assemblies and how to use the Windows Media Player primary interop assembly with your Visual Studio .NET solution.
About .NET Framework Data Types | Provides the information you need to translate the script-oriented Object Model Reference into Microsoft .NET Framework base data types.
Using the Player Control with Microsoft Visual Studio .NET | Provides details about how to add the Windows Media Player 9 Series control to your Visual Studio .NET solution.
Embedding the Player Control in a C# Solution | Provides step-by-step instructions for how to create a very simple C# application using the Player control.

See Also
- Player Control Guide
definitions (as metadata) of types implemented by a COM component. Only the publisher of a type library can produce a true PIA, which becomes the unit of official type definitions for interoperating with the underlying COM types.

For more information, see Primary Interop Assemblies (PIAs) on the MSDN Web site.

The following topics discuss how to register and re-distribute a PIA.

- Registering Primary Interop Assemblies for Application Development
- Re-distributing a Primary Interop Assembly with Your Application

See Also

- Embedding the Player Control in a .NET Framework Solution

Registering Primary Interop Assemblies for Application Development

You must register and install the Windows Media Player PIA by using the Assembly Registration Tool (RegAsm.exe) and the Global Assembly Cache Utility (GacUtil.exe).

To register a PIA, at the command prompt, type:

```
regasm assemblyname
```

In this command, `assemblyname` is the file name of the assembly to be registered.

The following example registers the `wmppia.dll` PIA.

```
regasm C:\WMSDK\WMPSDK9\redist\wmppia.dll
```

Regasm.exe adds a registry entry for the PIA under the same registry key as the original type library.

After you register the PIA, install it in the global assembly cache (GAC) using the Global Assembly Cache Utility.

To install a PIA in the GAC, at the command prompt, type:

```
Gacutil /i assemblyname
```

In this command, `assemblyname` is the file name of the assembly to be installed.
The following example installs the Windows Media Player PIA in the GAC

```
Gacutil /i C:\WMSDK\WMPSDK9\redist\wmppia.dll
```

**Note:** You should search your computer for both RegAsm.exe and Gacutil.exe. Usually these can be found in \%windir\%\Microsoft.NET\Framework\v1.0.xxxx where xxxx is the build number of the .NET Framework release you are using, or in the \FrameworkSDK\Bin folder where you installed the development environment.

The folder that contains the C# sample application included with the full installation of this SDK also contains a script file named regpiagac.vbs. This utility can make it easy to register the PIA and add it to the GAC. Simply double-click the file and provide the correct paths when prompted to complete the process. If you installed this SDK and Visual Studio .NET using the default installation paths, you should only need to change drive letters. If you did not install the products using the default paths, unexpected results may occur.

**See Also**

- [About Primary Interop Assemblies](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

Re-distributing a Primary Interop Assembly with Your Application

Primary interop assemblies (PIAs) are redistributed to end users as part of a .NET Framework application. Other than requiring that each relevant COM type library be registered on a user's computer, the deployment of an application containing one or more PIAs is the same as for any .NET-based application.

By definition, PIAs are always signed by their publisher to ensure uniqueness. As the single, official definition of the types they describe, you can expect popular PIAs to be installed in the global assembly cache even if you deploy the same assembly to an application directory. The common language runtime always directs your application to the PIA in the global assembly cache when both the global assembly and your local assembly have the same vendor signature. In this case, your application is not protected from version changes initiated by the vendor. The best practice is to always install the latest version from the vendor. However, when your application requires such protection, you can generate your own interop assembly by using Type Library Importer (Tlbimp.exe) instead of using the PIA.

**See Also**

- [About Primary Interop Assemblies](#)
About .NET Framework Data Types

This section contains the information you need to translate the script-oriented Object Model Reference into Microsoft .NET Framework base data types. The Windows Media Player script reference has almost all the information you need to use the Windows Media Player control in a .NET Framework-based program, and in most cases, the syntax will be similar to other scripting languages such as Microsoft JScript.

The Player reference provides the JScript data type and if necessary, the C++ conversion. For example, a Number might be returned by a method. JScript treats all numbers the same, but other languages distinguish between different types of numbers (integer, floating point, and so on). The reference gives the C++ conversion for number data types because numbers can be processed differently by C++; for example, C++ uses the int data type for integer arithmetic and float for floating point.

The .NET Framework uses a slightly different system of base data types. The following table shows the differences in the common data types you are likely to use. For more information on .NET Framework base data types and the conversion to other data type systems, see the .NET Framework Developer's Guide discussion of System Namespace base data types. The table gives the .NET Framework class name and the C# data type; data types for other languages are defined for each language in their respective language references.

<table>
<thead>
<tr>
<th>Script data type</th>
<th>C++ data type</th>
<th>.NET framework class (C# data type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>int</td>
<td>Int32 (int)</td>
</tr>
<tr>
<td>Number</td>
<td>long</td>
<td>Int32 (int)</td>
</tr>
<tr>
<td>Number</td>
<td>double</td>
<td>Double (double)</td>
</tr>
<tr>
<td>Number</td>
<td>float</td>
<td>Single (float)</td>
</tr>
<tr>
<td>String</td>
<td>BSTR</td>
<td>String (string)</td>
</tr>
<tr>
<td>Boolean</td>
<td>VARIANT_BOOL</td>
<td>Boolean (bool)</td>
</tr>
<tr>
<td>Object</td>
<td>Object</td>
<td>Object (object)</td>
</tr>
</tbody>
</table>

If you are using Microsoft Visual Studio .NET, you can use the Microsoft IntelliSense® technology to determine what data type is expected for a specific function. For example, when using Microsoft Visual C# .NET you can type the following code fragment:

```csharp
playercontrolname.settings.setMode(
```

The IntelliSense popup will tell you that the **setMode** method requires two parameters, a **string** and a **bool**.

See Also
Using the Player Control with Microsoft Visual Studio .NET

You can use the Windows Media Player ActiveX control in a .NET Framework application very easily by using the following procedure:

1. Make sure that the latest version of the Player and the Player SDK is installed on your machine.
2. Install and register the PIA. For more information, see Registering Primary Interop Assemblies for Application Development.
3. Start a new project in Visual Studio .NET.
4. From the Tools menu, select Customize Toolbox.
5. In the COM Components tab, click on the check box for Windows Media Player item to select the Player. Click OK to close the dialog and add the Player to the Toolbox.

Once you have added the Player to the Toolbox, it will be listed in the Components tab. You can select the Player in the toolbox and add it to a form. Visual Studio .NET will give the Player a default name such as "axWindowsMediaPlayer1". Adding the Player to the Toolbox also adds references to two libraries which it creates. You can see them in the Solution Explorer under references and they will have names similar to: "AxWMPLib" and "WMPLib".

Visual Studio will automatically recognize that you are adding a reference to a type library that has a registered PIA, so it will use the registered PIA instead of reimporting the type library.

In order to use all of the objects in the PIA, you must reference the PIA namespace as shown in the following code snippet:

```csharp
using Microsoft.MediaPlayer.Interop;
```

When you first add the Player control to a form, it will be visible. Unless you want to use the visible image of the Player in your application, you can hide the default Player one of the following code snippets:

```csharp
axWindowsMediaPlayer1.uiMode = "invisible";
```

or

```csharp
axWindowsMediaPlayer1.visible = "false";
```
You can also set the **Size** property to 0,0 in the **Properties** window by clicking on the visible image of the Player in the **Form Design** mode.

See Also

- [Embedding the Player Control in a .NET Framework Solution](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Embedding the Player Control in a C# Solution**

You can use the functionality of the Player in a C# application very easily. Install the Player and add it to the Microsoft Visual Studio .NET Toolbox using the procedure discussed in [Using the Player Control with Microsoft Visual Studio .NET](#). The following procedure will create an application that plays video and has a play and stop button. Before you begin, you should make sure your PIA is registered and installed in the global assembly cache. For more information, see [Registering Primary Interop Assemblies for Application Development](#).

**Add the Video Window**

Once you have added the Player to a form, drag the window where you want the video window of the Player to appear. With the Player control selected in the form, change the **uiMode** property to none to hide the UI controls. If you are playing a video, the video will appear in the window; otherwise a visualization will appear.

**Add Two Buttons and Adjust the Form**

Then add two buttons to your form. Select the first one and in the Properties pane, change the Text property to "Play". Select the second button and change its Text property to "Stop". Position the two buttons and the video window on the form and size the form accordingly. You can name the form by selecting it and changing the Text property to something like "My Player".

**Add the Play Code**

Double-click the Play button to reveal the Code window. In C#, the following code will be displayed:

```csharp
private void button1_Click(object sender, System.EventArgs e)
{
}
```

Add this line between the two curly braces:
axWindowsMediaPlayer1.URL = "c:\\mediafile.wma";

In the preceding code example, axWindowsMediaPlayer1 is the name of the Player control and mediafile.wma is the name of the media you want to play. Any valid file path can be used. Note that in C#, you must use double back slashes to escape any back slashes in your URL. If you have the media from the Player SDK added to your media library, you can use this line instead:

axWindowsMediaPlayer1.currentPlaylist = axWindowsMediaPlayer1.mediaCollection.getByName("l

Because the autoStart property is true, the Player will start playing when you set a current playlist or set the URL property.

Add the Stop Code

Double-click the Stop button to reveal the Code Window. In C#, the following code will be displayed:

```csharp
private void button2_Click(object sender, System.EventArgs e)
{
}
```

Add this line between the two curly braces:

```csharp
axWindowsMediaPlayer1.controls.stop();
```

Build your Application

You are now ready to build your application. From the Build menu, choose Build Solution. If all went well, you will have created an application. If you distribute your application, be sure to distribute AxInterop.WMPLib.dll and wmppia.dll as well. You will also need to make sure that the Player is installed on any machine that your application runs on.

See Also

- [Embedding the Player Control in a .NET Framework Solution](#)
Player 6.4 object model, so you need to understand how to update your existing code if you want to support Windows Media Player 7 and later in your projects.

For a sample that demonstrates how to detect the user's Windows Media Player version and current browser, see the sample named "detection" that is installed with this SDK. For more information about samples, see Samples.

This guide refers to the new object model as the Windows Media Player 9 Series object model. For details about which functionality is available in each version of Windows Media Player, see Object Model Reference.

These object model issues are covered by the following topics:

- Differences Between the Object Models
- Using the Windows Media Player 9 Series Object Model
- Error Handling
- Playlists
- Closed Captioning
- DVD
- Detailed Object Model Comparison

See Also

- Player Control Guide

Differences Between the Object Models

There are two primary differences between the Windows Media Player 6.4 object model and the Windows Media Player 9 Series object model.

- **CLSID** The Windows Media Player 9 Series object model is a complete departure from the version 6.4 object model. The Component Object Model (COM) specification requires that all existing interfaces must continue to function in new versions of a COM component. This means that new interfaces can be added to a COM component, but existing interfaces must never be altered, ensuring older client code will always work with the particular component it was designed to use. Therefore, the Windows Media Player 7 or later ActiveX control has a new class ID: 6BF52A52-394A-11D3-B153-00C04F79FAA6. If you want to take advantage of the new functionality of the control for this version, you must change your CLSID.

- **Hierarchical Object Model** If you've been using the Windows Media Player 6.4 ActiveX control, you may have noticed that all the properties, methods, and events are accessed through the same object: the **Player** object. By contrast, the Windows Media Player 9 Series object model is organized as a hierarchy...
of objects. The **Player** object is still the root object, but functions are now accessed through a variety of child objects.

**See Also**

- **Object Model Migration Guide**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

## Using the Windows Media Player 9 Series Object Model

Most of the tasks you may have been performing using the Windows Media Player 6.4 ActiveX control object model will require a new approach. In many cases, the names of the properties, methods, and events have changed in the Windows Media Player 9 Series object model. For instance, to specify the file path in the version 6.4 object model, you set the `Player6.FileName` property:

```javascript
WMP64.FileName = "http://www.microsoft.com/somefile.wmv";
```

Using the Windows Media Player 9 Series object model, you must set the `Player.URL` property:

```javascript
WMP9.URL = "http://www.microsoft.com/somefile.wmv";
```

Alternatively, using the 9 Series object model, you can obtain a **Media** object from **Media Library**, and then set the **Player.currentMedia** property:

```javascript
// Get the first media object in the media collection.
var MyMediaItem = WMP9.mediaCollection.getAll().item(0)

// Make the MyMediaItem object the current media.
WMP9.currentMedia = MyMediaItem;
```

Much of the functionality in the Windows Media Player 9 Series object model is accessed through the object hierarchy. As the previous example showed, you can obtain a **Playlist** object by using the `getAll` method of the **mediaCollection** object, which is accessed through the root **Player** object. You can then obtain a particular **Media** object from the **Playlist** object by using the `item` method of the **Playlist** object. There are five additional methods accessible through the **mediaCollection** object that return a **Playlist** object; each method allows you to retrieve the object based on specific criteria, like genre or album.

The hierarchical structure of the Windows Media Player 9 Series ActiveX control object model provides a more logical approach to organizing the properties, methods, and events available for your use. All the functionality for the Player controls is contained in the **Controls** object, all the functionality for the Player network
connection is contained in the Network object, and so forth. For example, to start content playing using the version 6.4 object model, you use the Player6.\texttt{Play} method:

\texttt{WMP64.Play();}

Using the 9 Series object model, you must access the \texttt{Play} method using the \texttt{Controls} object:

\texttt{WMP9.controls.play();}

The depth of the object model, however, can lead to very long script statements:

\texttt{WMP9.currentPlaylist.appendItem(WMP9.mediaCollection.getByName("MySong").item(0));}

Statements like the preceding one can be made much simpler and more readable by working with individual named objects. The following example replaces the preceding code statement with syntax using separate object variables:

```
// Store the current playlist object.
var pl = WMP9.currentPlaylist;

// Get a playlist from the media collection that contains
// one media item named "MySong".
var temp = WMP9.mediaCollection.getByName("MySong");

// Get the individual media item from the temp playlist.
var song = temp.item(0);

// Append the media item to the current playlist.
pl.appendItem(song);
```

This coding style requires more lines of script, but is much easier to follow, especially with the added comments. There is another advantage: the \texttt{currentPlaylist} object is easy to reuse because it is stored in the variable \texttt{pl}.

Many of the properties, methods, and events in the Windows Media Player 9 Series object model set or retrieve different values, or return values of a different type or number, compared to corresponding functionality in the version 6.4 object model. For instance, when \texttt{Player6.openState} retrieves 2, that value corresponds to the Visual Basic constant \texttt{nsLoadingNSC}, meaning the Player is loading a station file with a .nsc file name extension. But when the 9 Series object model property \texttt{Player.openState} retrieves 2, that value corresponds to the state PlaylistLocating, meaning Windows Media Player is attempting to locate a playlist. Additionally, the \texttt{Player6.openState} property can retrieve seven different values, while the 9 Series \texttt{Player.openState} property can retrieve 22 different values. Be sure to refer to the Object Model Reference section of the Windows Media Player SDK when revising code to use a different object model version.

See Also

- About the Player Object Model
- Object Model Reference
- Object Model Migration Guide
Error Handling

The Windows Media Player 6.4 ActiveX control provides default error handling by displaying error messages in dialog boxes and on the status bar. You can also provide custom error handling by processing errors in your script. Error handling is event driven, which means you receive a notification for each error, and must decide how to deal with each error event when it occurs. For more information about handling errors using the version 6.4 object model, see the Error Handling section of the Version 6.4 Player Object Model Guide, which is part of the Windows Media Player SDK.

The Windows Media Player 9 Series object model provides the Error object and the ErrorItem object to handle errors. These two objects work together to provide you with an error handling mechanism that gives you complete and flexible control of the error handling process. The Error object provides access to a collection of ErrorItem objects; each ErrorItem object provides details about an individual error message.

When an error occurs, the error information is posted to an error queue. The queue is a collection of ErrorItem objects. As each error is added to the queue, it is associated with an index number (beginning with zero) that can be used to identify the particular ErrorItem object. The Error.errorCount property retrieves the number of errors in the error queue. Since the index numbers are zero-based, the most recent error posted to the queue will always have an index value equal to Error.errorCount minus one.

You can create an error event handler for Windows Media Player using script. The following JScript example shows how to retrieve the most recent error item from the error queue and display the error code and error description using the 9 Series object model. The Player object was created with ID = "WMP9".

```html
<!-- Create an error event handler for Windows Media Player 9 Series errors. -->
<SCRIPT LANGUAGE = "JScript" FOR = WMP9 EVENT = error()>

// Store the number of errors in the error queue.
var max = WMP9.error.errorCount;

// Retrieve most recent ErrorItem object.
var err = WMP9.error.item(max-1)

// Store the error code number.
var errNum = err.errorCode;

// Store the error description string.
var errDesc = err.errorDescription;

// Build a message string to notify the user.
var msg = "Error number: " + errNum + "\n";
msg += "Error description: " + errDesc;

// Display the message box.
alert(msg);

</SCRIPT>
```
The **Error** object has two additional methods that you can use. The `Error.clearErrorQueue` method allows you to remove all the errors from the error queue and reset the index number to zero. You have complete control over this process; you can keep errors in the queue for as long as you need them to be available, and then empty the queue when you're finished handling the errors.

The `Error.webHelp` method provides a way to display the most current error information to the user by using the Internet. When called, this method transfers all relevant information about the first error in the queue (the one with index zero) to Microsoft Windows Media Player Web Help, which displays further information about the error in the current browser window.

**See Also**

- **Error Object**
- **ErrorItem Object**
- **Object Model Migration Guide**

### Playlists

The Windows Media Player 6.4 ActiveX control object model includes four methods and one property for working with Windows Media metafile playlists:

- `Player6.GetCurrentEntry`
- `Player6.SetCurrentEntry`
- `Player6.GetMediaParameter`
- `Player6.GetMediaParameterName`
- `Player6.EntryCount`

Together, these provide limited functionality for navigating a playlist metafile with a .asx file name extension and retrieving information about the entries contained in the playlist.

Windows Media Player 7 introduced **Media Library**. **Media Library** allows users to organize their digital media content, as well as to create custom playlists that can be managed from the Player graphical user interface. The Windows Media Player 9 Series ActiveX control object model provides support for working with Media Library playlists, as well as playlists contained in Windows Media metafiles with an .asx file name extension.

**Note**  For security reasons, the user must grant access rights to **Media Library** before your program can manipulate its content. Access rights can only be requested and granted through the Windows Media Player 9 Series object model. For details about access rights, see **Media Library Access**.
The 9 Series object model includes three objects for handling playlists. The PlaylistCollection object provides functionality for organizing playlists; it represents the entire collection of playlists in the user's Media Library. The PlaylistArray object provides a way to retrieve a specific playlist from the PlaylistCollection object by using an index number; two of the PlaylistCollection object methods retrieve a PlaylistArray object. The Playlist object provides the properties and methods necessary to manipulate media items contained in a single playlist.

As an example, since each playlist in Media Library has a unique name, you can retrieve a playlist from Media Library using the PlaylistCollection.getByName method:

```java
// Retrieve a PlaylistArray object that contains
// exactly one Playlist object.
var plarray = WMP9.playlistCollection.getByName("MyPlaylist");

// Get the Playlist object from the PlaylistArray object.
// The Playlist object has index number zero.
var pl = plarray.item(0);

// Make the retrieved playlist the current playlist.
WMP9.currentPlaylist = pl;
```

You will most frequently want to work with the current playlist. While it is possible to use several playlist objects, only one can be retrieved by the Player.currentPlaylist property at any given time: the one that Windows Media Player is processing at that moment.

When Windows Media Player 9 Series plays a Windows Media metafile with a .asx file name extension, it first creates a Playlist object. Next, it fills the object with the information from the .asx playlist, and then makes that Playlist object the current playlist. This means that you can use the properties and methods associated with the Playlist object to manipulate .asx playlists exactly as you would handle playlists in Media Library. For instance, to retrieve the number of entries in an .asx playlist using the version 6.4 object model, you use the Player6.EntryCount property:

```java
var entrycount = WMP64.EntryCount;
```

Using the Windows Media Player 9 Series object model, you use the Playlist.count property:

```java
var entrycount = WMP9.currentPlaylist.count;
```

Using the version 6.4 control, you can use the Player6.GetCurrentEntry method to retrieve the index of the current entry in a .asx playlist:

```java
var entrynum = WMP64.GetCurrentEntry();
```

You can achieve the same result using the 9 Series object model in script. The following JScript example compares the current media object to each item in the playlist. When Media.isIdentical returns true, a message box displays the index of the current media item.

```javascript
function matchit(){
  // Store the current playlist object in a variable.
  var pl = WMP9.currentPlaylist;

  // Loop through the playlist one entry at a time.
  for (var i = 0; i < pl.count; i++){
    // Test whether the current media item matches
    // the item in the playlist at the current loop index.
```
if (WMP9.currentmedia.isIdentical(pl.item(i))){

    // They match, display the index.
    var message = "Current media at index: " + i;
    alert(message);

    // Exit the function, don't continue looping!
    return;
}
}
}

To specify the index of the current entry in a .asx playlist, you use Player6.SetCurrentEntry. Playlist entry indexes in version 6.4 start with 1, so to make the second entry in a metafile playlist the current one, use the following syntax:

WMP6.SetCurrentEntry(2);

Windows Media Player 9 Series playlist entry indexes are zero-based; to make the second entry in a metafile playlist the current one using the 9 Series object model, use the following syntax:

WMP9.controls.currentItem = WMP9.currentPlaylist.item(1);

The following JScript example demonstrates a function that accepts an index number as a parameter, and then makes the playlist entry that corresponds to the index the current media item:

function setindex(idx){
    // Store the current playlist in a variable.
    var pl = WMP9.currentPlaylist;

    // Get the first playlist entry.
    var firstmedia = pl.item(0);

    // Start the Player to allow navigation within the playlist.
    WMP9.controls.playItem(firstmedia);

    // Test whether idx is within a valid range.
    if (idx < pl.count && idx >= 0){

        // Set the currentItem to the desired playlist item.
        WMP9.controls.currentItem = pl.item(idx);

        // Display the name of the media item.
        alert(WMP9.currentMedia.name);
        return true;
    }

    // The index is out of range, stop the Player, alert the user.
    WMP9.controls.stop();
    alert("Index out of range");
    return false;
}

Windows Media metafiles can contain custom parameter elements, which you specify by using the <PARAM> tag. Using the version 6.4 object model, you can retrieve the name of a particular parameter with the Player6.GetMediaParameterName method. The following JScript example retrieves the name of the first parameter in the first entry of a .asx playlist:

var paramname = WMP6.GetMediaParameterName(1,1);
Similarly, you can retrieve the value associated with the parameter using `Player6.GetMediaParameter`:

```javascript
var paramvalue = WMP6.GetMediaParameter(1, paramname);
```

The following JScript example uses the Windows Media Player 9 Series object model to retrieve the parameter name and value from the first entry in a .asx playlist:

```javascript
function getattribute(){
    // Store the first playlist entry as a Media object.
    var firstmedia = WMP9.currentplaylist.item(0);

    // Get the name of the first parameter in the object named firstmedia.
    var attname = firstmedia.getAttributeName(0);

    // Get the value of the first parameter in the object named firstmedia.
    var attval = firstmedia.getItemInfo(attname);

    // Display the information.
    alert(attname + ": " + attval);
}
```

You can use the `PlaylistCollection.importPlaylist` method to add a .asx playlist to `Media Library`. Once imported, the metafile playlist becomes a `Media Library` playlist, so you can manipulate it using all the properties and methods at your disposal. The user must grant full access rights to `Media Library` in order for your application to be able to use the `importPlaylist` method.

You can use `PlaylistCollection.getByName` to test whether a playlist exists. This method always returns a valid `PlaylistArray` object. If the playlist array retrieved contains exactly one playlist, then there exists a playlist with that name in `Media Library`. Otherwise, the playlist array will contain no playlist object; this means there is no playlist in `Media Library` with the name passed as an argument to the `getByName` method. The following JScript example demonstrates this:

```javascript
// Specify a .asx playlist file.

// Open the playlist and start playing the content.
WMP9.controls.play();

// Store the current playlist object.
var pl = WMP9.currentPlaylist;

// Attempt to retrieve from Media Library
// a playlist having the same name as the current playlist.
var plarray = WMP9.playlistCollection.getByName(pl.name);

// Test whether the PlaylistArray object, plarray, contains
// a Playlist object.
if (!plarray.count)
    // If plarray contains no playlist, then import
    // the current one.
    WMP9.playlistCollection.importPlaylist(pl);
}
```

See Also

- [Managing Playlists](#)
- [Object Model Migration Guide](#)
Closed Captioning

The Windows Media Player 6.4 ActiveX control includes an integrated closed caption display panel that, when made visible, enables Synchronized Accessible Media Interchange (SAMI) closed captions and displays the closed caption text. The 9 Series control enables SAMI closed caption display by using an HTML `<DIV>` element. For example:

```html
<DIV ID = "CCDiv"></DIV>
```

This technique provides you with complete flexibility, since you can design your Web page to display closed captions in a customized manner; the closed caption display is no longer required to be in a fixed location adjacent to the Windows Media Player user interface.

Once you have created an area to display closed captions, use the `ClosedCaption.captioningID` property to specify the location where Windows Media Player renders the closed caption text.

```javascript
Player.closedCaption.captioningID = "CCDiv";
```

The Windows Media Player Software Development Kit (SDK) contains a working sample of an embedded Player that displays closed caption text. To get the SDK samples, download and install the complete Windows Media Player SDK from the Microsoft Web Site. For more information about closed captions and SAMI, see the [Microsoft Accessibility Web site](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mfc90/html/mfc90__closedcaptioning.asp).

See Also

- Adding Closed Captions to Digital Media
- `ClosedCaption` Object
- `Object Model Migration Guide`
DVD

The Windows Media Player 6.4 ActiveX control contains a DVD object that exposes a variety of methods and properties, and one event, for dealing specifically with DVD content. For instance, to determine the number of DVD titles available, you use the `Player6.titlesAvailable` property:

```javascript
var numTitles = WMP64.DVD.TitlesAvailable;
```

The Windows Media Player 9 Series object model implements a more integrated approach to DVD. DVD-specific properties, methods, and events are implemented only where needed. Otherwise, existing object model functionality works with DVD media as you might expect. For example, to determine the number of DVD titles available using 9 Series, you retrieve a Playlist object from the Cdrom object:

```javascript
var dvdTitles = WMP9.cdromCollection.item(0).playlist;
```

The preceding example retrieves a Playlist object with a first entry representing the DVD media in the first drive. Additional entries represent individual DVD titles. Therefore, the number of titles available can be calculated as follows:

```javascript
var numTitles = dvdTitles.count - 1;
```

Here are the main differences to keep in mind when migrating from version 6.4:

- DVD playback is supported only when using Windows Media Player for Windows XP or later and the Windows XP operating system or later.
- DVD-ROM drives are enumerated exactly like CD-ROM drives using the `CdromCollection` object.
- Individual DVD-ROM drives are managed using the `Cdrom` object.
- The DVD object extends the object model with methods and one property specifically for working with DVD.
- There are two new events, `OpenPlaylistSwitch` and `DomainChange`, specifically for working with DVD.
- DVD content is organized using Playlist objects and Media objects.
- DVD languages are managed using the language functionality available from the Controls object (Windows Media Player 9 Series or later).
- Transport controls and position information for DVD work the same as for other digital media types.

For a description of a DVD Player application written in Visual Basic 6.0 using Windows Media Player 9 Series SDK, see Using Windows Media Player with Visual Basic.

See Also

- Object Model Migration Guide
- Using Windows Media Player with Visual Basic
### Detailed Object Model Comparison

The following table compares the Windows Media Player 6.4 object model properties with the Windows Media Player 9 Series object model.

<table>
<thead>
<tr>
<th>Windows Media Player 6.4 property</th>
<th>Windows Media Player 9 Series equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player6.AllowChangeDisplaySize</td>
<td>The Windows Media Player 9 Series display automatically resizes to fit the media. You can set the height and width properties in the <code>&lt;OBJECT&gt;</code> tag or in script.</td>
</tr>
<tr>
<td>Player6.AllowScan</td>
<td>Controls.fastForward and Controls.fastReverse are automatically enabled for file types that support these methods.</td>
</tr>
<tr>
<td>Player6.AnglesAvailable</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6.AnimationAtStart</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6.AudioStream</td>
<td>Use Controls.currentAudioLanguageIndex.</td>
</tr>
<tr>
<td>Player6.AudioStreamsAvailable</td>
<td>Use Controls.audioLanguageCount.</td>
</tr>
<tr>
<td>Player6.AutoRewind</td>
<td>Use Controls.currentPosition in script to specify or retrieve the current position. Alternatively, use markers and the Player.markerHit event.</td>
</tr>
<tr>
<td>Player6.AutoSize</td>
<td>Automatic sizing is the default behavior. To override automatic sizing, set the height and width properties in the <code>&lt;OBJECT&gt;</code> tag or in script.</td>
</tr>
<tr>
<td>Player6.AutoStart</td>
<td>Use Settings.autoStart.</td>
</tr>
<tr>
<td>Player6.BaseURL</td>
<td>Use Settings.baseURL.</td>
</tr>
</tbody>
</table>
Player6.CanPreview Not available.

Player6.CanScan Use Controls.isAvailable("FastForward") and Controls.isAvailable("FastReverse").

Player6.CanSeek Use Controls.isAvailable to test whether a particular seek method can be performed.

Player6.CanSeekToMarkers Use Controls.isAvailable("CurrentMarker"). Use Media.markerCount to retrieve the count of markers in a particular media item. Use Controls.currentMarker to specify or retrieve the current marker number.

Player6.CaptioningID Use ClosedCaption.captioningID.

Player6.CCActive Not available. See Closed Captioning for information about how closed captioning has changed in Windows Media Player.

Player6.ChannelDescription Not available.

Player6.ChannelName Not available.

Player6.ChannelURL Not available.

Player6.ClickToPlay Not available. You should provide controls in your user interface to start playback. Alternatively, the user can right-click the video image to open a pop-up menu that contains a Play/Pause selection if Player.enableContextMenu equals true.

Player6.ClientID Not available.

Windows Media Player 9 Series allows the user to select whether a unique Player ID is transmitted to content providers.

If the user selects this option, the Player sends a unique ID to the Windows Media server. The ID is logged in the server's log file, located in the..\system32\logfiles folder by default. The log field name is "c-playerid". Server logging is not enabled by default in Windows Media Services.

If the user does not select this option, the server generates a random session ID, which is unique for each client for a given session.

For more information, see the Windows Media Services 9 Series documentation.

Player6.CodecCount Not available.

Player6.ColorKey Not available.

Player6.ConnectionSpeed Not available. Use Network.bitRate to determine the current bit rate.
Player6.ContactAddress Not available.
Player6.ContactEmail Not available.
Player6.ContactPhone Not available.

Player6.CreationDate
Use MediaCollection.getMediaAtom("CreationDate") to retrieve the index of the creation date atom. Use Media.getItemInfoByAtom to retrieve the metadata.


Player6.CurrentAudioStream
Use Controls.currentAudioLanguageIndex.

Player6.CurrentButton Not available.

Player6.CurrentCCService Not available.

Player6.CurrentChapter
Retrieve the current playlist. If the current playlist is not the same one as the playlist returned by Cdrom.playlist, then there is no current chapter. Otherwise, the current chapter number is the index of the current media in the current playlist.

Player6.CurrentDiscSide Not available.

Player6.CurrentDomain
Use DVD.domain.

Player6.CurrentMarker
Use Controls.currentMarker.

Player6.CurrentPosition
Use Controls.currentPosition.


Player6.CurrentTime
Use Controls.currentPositionTimeCode, Controls.currentPositionString, or Controls.currentPosition.

Player6.CurrentTitle
Retrieve the current playlist. If the current playlist is the same one as the playlist returned by Cdrom.playlist, then the title number is the index of the current media in the current playlist.

Player6.CurrentVolume Not available.

Player6.CursorType Not available. Use Internet Explorer styles instead.

Player6.DefaultFrame
Use Settings.defaultFrame, or use a <PARAM> attribute in the <OBJECT> element:

<PARAM NAME="defaultFrame" VALUE="right">

Player6.DisplayBackColor Not available.

Player6.DisplayForeColor Not available.

Player6.DisplayMode
The current position can be retrieved in seconds from the beginning as a Number using Controls.currentPosition, as a String formatted as HH:MM:SS (hours, minutes, seconds) using Controls.currentPositionString, or in time
code format using Controls currentPositionTimeCode.

Player6.DisplaySize

The default display automatically resizes to fit the media. You can set the height and width properties in the <OBJECT> tag, or in script. Use Player.fullScreen to switch to full-screen mode.

Player6.Duration

Use Media.duration.

Player6.DVD

Use Player.DVD.

Player6.EnableContextMenu

Use Player.enableContextMenu.

Player6.Enabled

Use Player.enabled.

Player6.EnableFullScreenControls

Using Windows Media Player 9 Series or later, full-screen controls are enabled automatically unless Player.uiMode = "none".

Player6.EnablePositionControls

Not available. You can provide custom controls or use Player.uimode to choose a default configuration.

Player6.EnableTracker

Not available. You can provide a custom control or use Player.uimode to choose a default configuration.

Player6.EntryCount

Use Playlist.count

Player6.ErrorCode

Use ErrorItem.errorCode.

Player6.ErrorCorrection

Not available.

Player6.ErrorDescription

Use ErrorItem.errorDescription.

Player6.FileName

Use Player.URL or Player.currentMedia. Use Controls.currentItem when working within a playlist.

Player6.FramesPerSecond

Not available.

Player6.HasError

Use Error.errorCount.

Player6.HasMultipleItems

Not available.

Player6.ImageSourceHeight

Use Media.imageSourceHeight.

Player6.ImageSourceWidth

Use Media.imageSourceWidth.

Player6.InvokeURLs

Use Settings.invokeURLs.

Player6.IsBroadcast

Use Network.sourceProtocol.

Player6.IsDurationValid

Not available. Media.duration contains a valid value when used with the current media object.

Player6.Language

Use Controls.currentAudioLanguage

Player6.LostPackets

Use Network.lostPackets.

Player6.MarkerCount

Use Media.markerCount.

Player6.Mute

Use Settings.mute.
Player6.OpenState
Use Player.openState.

Player6.PlayCount
Use Settings.playCount.

Player6.PlayState
Use Player.playState.

Player6.PreviewMode
Not available. Use a script loop structure with an HTML timer to duplicate this functionality.

Player6.Rate
Use Settings.rate.

Player6.ReadyState
Use Player.openState.

Player6.ReceivedPackets
Use Network.receivedPackets.

Player6.ReceptionQuality
Use Network.receptionQuality.

Player6RecoveredPackets
Use Network.recoveredPackets.

Player6.Root
Not available.

Player6.SAMIFilename
Use ClosedCaption.SAMIFilename.

Player6.SAMILang
Use ClosedCaption.SAMILang.

Player6.SAMISStyle
Use ClosedCaption.SAMISStyle.

Player6.SelectionEnd
Use Media.duration to determine the length of a Media object. Use a marker with Controls.currentMarker to specify a custom end position.

Player6.SelectionStart
Use Controls.currentPosition to start playback from a particular position or use a marker with Controls.currentMarker to specify a custom start position.

Player6.SendErrorEvents
Errors are queued. Use the Error object and the ErrorItem object to retrieve error information.

Player6.SendKeyboardEvents
Not available.

Player6.SendMouseClickEvents
Not available.

Player6.SendMouseMoveEvents
Not available.

Player6.SendOpenStateChangeEvents
Not available.

Player6.SendPlayStateChangeEvents
Not available.

Player6.SendWarningEvents
Not available.

Player6.ShowAudioControls
Not available. You can provide custom controls or use Player.uimode to choose a default configuration.

Player6.ShowCaptioning
Not available. You can provide a custom closed caption display.

Player6.ShowControls
Not available. You can provide custom controls or use Player.uimode to choose a default configuration.

Player6.ShowDisplay
Not available.
The following table compares the Windows Media Player version 6.4 object model methods with the Windows Media Player 9 Series object model.

<table>
<thead>
<tr>
<th>Player6 method</th>
<th>Windows Media Player 9 Series equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player6.ShowGotoBar</td>
<td>Not available. You can provide custom functionality using the Media object</td>
</tr>
<tr>
<td>Player6.ShowPositionControls</td>
<td>Not available. You can provide custom controls or use Player.uiMode to choose a default configuration.</td>
</tr>
<tr>
<td>Player6.ShowStatusBar</td>
<td>Not available. You can provide custom controls or use Player.uiMode to choose a default configuration.</td>
</tr>
<tr>
<td>Player6.ShowTracker</td>
<td>Not available. You can provide custom controls or use Player.uiMode to choose a default configuration.</td>
</tr>
<tr>
<td>Player6.SourceLink</td>
<td>Use Media.sourceURL.</td>
</tr>
<tr>
<td>Player6.StreamCount</td>
<td>Not available. Use Controls.audioLanguageCount to retrieve the number of audio language streams.</td>
</tr>
<tr>
<td>Player6.SubpictureStreamsAvailable</td>
<td>Not available</td>
</tr>
<tr>
<td>Player6.TitlesAvailable</td>
<td>Use the following: Player.Cdrom.playlist.count - 1</td>
</tr>
<tr>
<td>Player6.TotalTitleTime</td>
<td>Use currentMedia.duration or currentMedia.durationString.</td>
</tr>
<tr>
<td>Player6.TransparentAtStart</td>
<td>Use script to specify the height and width values to make the player visible or invisible.</td>
</tr>
<tr>
<td>Player6.UniqueID</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6ВидеоBorder3D</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6ВидеоBorderColor</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6ВидеоBorderWidth</td>
<td>Not available.</td>
</tr>
<tr>
<td>Player6.VolumesAvailable</td>
<td>Not available.</td>
</tr>
</tbody>
</table>
- **Player6.Cancel** Not available.
- **Player6.ChapterPlay** If already playing the specified title playlist, retrieve the desired chapter as a media object using the following syntax:

  ```javascript
  var media = Player.currentPlaylist.item(index);
  ```

  Then, specify `Player.currentMedia` using the media object returned.
- **Player6.ChapterPlayAutoStop** Not available.
- **Player6.ChapterSearch** If already playing the specified title playlist, retrieve the desired chapter as a media object using the following syntax:

  ```javascript
  var media = Player.currentPlaylist.item(index);
  ```

  Then, specify `Player.currentMedia` using the media object returned.
- **Player6.FastForward** Use `Controls.fastForward`.
- **Player6.FastReverse** Use `Controls.fastReverse`.
- **Player6.ForwardScan** Use `Settings.rate`.
- **Player6.GetAllGPRMs** Not available.
- **Player6.GetAllSPRMs** Not available.
- **Player6.GetAudioLanguage** Use `Controls.currentAudioLanguage` to retrieve the LCID of the current audio language.
- **Player6.GetCodecDescription** Not available.
- **Player6.GetCodecInstalled** Not available.
- **Player6.GetCodecURL** Use `ErrorItem.customUrl`.
- **Player6.GetCurrentEntry** Use script to loop through the current playlist. Use `Media.isIdentical` to compare each entry in the playlist to the `Player.currentMedia` object.
- **Player6.GetMarkerName** Use `Media.getMarkerName`.
- **Player6.GetMediaInfoString** Use `Media.getItemInfo`, `Media.getItemInfoByAtom`, and their associated methods to retrieve metadata.
- **Player6.GetMediaParameter** Use `Playlist.item` to retrieve a media item. Then use `Media.getItemInfo` to retrieve the parameter string.
- **Player6.GetMediaParameterName** Use `Playlist.item` to retrieve a media item. Then use `Media.getAttributeName` to retrieve the parameter string.
Player6.GetNumberOfChapters
If a title is currently playing, use `currentPlaylist.count`.

Player6.GetStreamGroup
Not available.

Player6.GetStreamName
Not available.

Player6.GetStreamSelected
Not available.

Player6.GetSubpictureLanguage
Not available.

Player6.GoUp
Use `DVD.back`.

Player6.IsSoundCardEnabled
Not available.

Player6.LeftButtonSelect
Not available.

Player6.LowerButtonSelect
Not available.

Player6.MenuCall
Use `DVD.titleMenu` or `DVD.topMenu`.

Player6.Next
Use `Controls.next`.

Player6.NextPGSearch
Use `Controls.next`.

Player6.Open
Use `Player.URL` or `Player.currentMedia`. Files always open asynchronously.

Player6.Pause
Use `Controls.pause`.

Player6.Play
Use `Controls.play`.

Player6.Previous
Use `Controls.previous`.

Player6.PrevPGSearch
Use `Controls.previous`.

Player6.ResumeFromMenu
Use `DVD.resume`.

Player6.RightButtonSelect
Not available.

Player6.SetCurrentEntry
Retrieve a media object using `currentPlaylist.item(entryNumber)`. Then, specify the retrieved media object using `Controls.currentItem`.

Player6.ShowDialog
Not available.

Player6.StillOff
Use `Controls.play`. Alternatively, use `Controls.Next` if currently in still mode.

Player6.Stop
Use `Controls.stop`.

Player6.StreamSelect
Not available. Use `Controls.currentAudioLanguage` to specify an audio language stream.

Player6.TimePlay
From the root playlist, use `currentPlaylist.item(index)` to retrieve a media object. Then, set the media object as the current one using `Controls.currentItem`. Then, specify `Controls.currentPosition` using a time value in seconds.

Player6.TimeSearch
Use `Controls.currentPosition`.

Player6.TitlePlay
If already playing the specified title playlist, retrieve the
desired chapter as a media object using the following syntax:

```javascript
var media = Player.currentPlaylist.item(index);
```

Then, specify `Player.currentMedia` using the media object returned.

Alternatively, use `currentPlaylist.item` to retrieve a media object, and then use the media object returned to specify `Controls.currentItem`.

### Windows Media Player 6.4 event

#### Player6Buffering

- **Windows Media Player 9 Series equivalent**: Use `Player.Buffering`.

#### Player6Click

- **Windows Media Player 9 Series equivalent**: Use `Player.Click`.

#### Player6DoubleClick

- **Windows Media Player 9 Series equivalent**: Use `PlayerDoubleClick`.

#### Player6Disconnect

- **Windows Media Player 9 Series equivalent**: Not available.

#### Player6DisplayModeChange

- **Windows Media Player 9 Series equivalent**: Not available.

#### Player6DVDNotify

- **Windows Media Player 9 Series equivalent**: `Player.DomainChange` and `Player.OpenPlaylistSwitch` are DVD-specific events. Other events related to playlists, media, and CD-ROM media may apply as well depending on the application.

#### Player6EndOfStream

- **Windows Media Player 9 Series equivalent**: Use `Player.PlayState`.

#### Player6Error

- **Windows Media Player 9 Series equivalent**: The event is unchanged. Errors, however, are queued. Use the `Error` object with the `ErrorItem` object to retrieve error information from the queue. See the example code in the preceding section, Error handling.

#### Player6KeyDown

- **Windows Media Player 9 Series equivalent**: Use `Player.Keydown`.

#### Player6KeyPress

- **Windows Media Player 9 Series equivalent**: Use `Player.KeyPress`.

#### Player6KeyUp

- **Windows Media Player 9 Series equivalent**: Use `Player.KeyUp`.

#### Player6MarkerHit

- **Windows Media Player 9 Series equivalent**: Use `Player.MarkerHit`.

#### Player6MouseDown

- **Windows Media Player 9 Series equivalent**: Use `Player.MouseDown`.

#### Player6MouseMove

- **Windows Media Player 9 Series equivalent**: Use `Player.MouseMove`.

#### Player6MouseUp

- **Windows Media Player 9 Series equivalent**: Use `Player.MouseUp`.

The following table compares the Windows Media Player version 6.4 object model events with the Windows Media Player 9 Series object model.

<table>
<thead>
<tr>
<th>Windows Media Player 6.4 event</th>
<th>Windows Media Player 9 Series equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player6Buffering</td>
<td>Use <code>Player.Buffering</code></td>
</tr>
<tr>
<td>Player6Click</td>
<td>Use <code>Player.Click</code></td>
</tr>
<tr>
<td>Player6DoubleClick</td>
<td>Use <code>PlayerDoubleClick</code></td>
</tr>
<tr>
<td>Player6Disconnect</td>
<td>Not available</td>
</tr>
<tr>
<td>Player6DisplayModeChange</td>
<td>Not available</td>
</tr>
<tr>
<td>Player6DVDNotify</td>
<td><code>Player DomainChange</code> and <code>Player OpenPlaylistSwitch</code> are DVD-specific events. Other events related to playlists, media, and CD-ROM media may apply as well depending on the application.</td>
</tr>
<tr>
<td>Player6EndOfStream</td>
<td>Use <code>Player.PlayState</code></td>
</tr>
<tr>
<td>Player6Error</td>
<td>The event is unchanged. Errors, however, are queued. Use the <code>Error</code> object with the <code>ErrorItem</code> object to retrieve error information from the queue. See the example code in the preceding section, Error handling.</td>
</tr>
<tr>
<td>Player6KeyDown</td>
<td>Use <code>Player.Keydown</code></td>
</tr>
<tr>
<td>Player6KeyPress</td>
<td>Use <code>Player.KeyPress</code></td>
</tr>
<tr>
<td>Player6KeyUp</td>
<td>Use <code>Player.KeyUp</code></td>
</tr>
<tr>
<td>Player6MarkerHit</td>
<td>Use <code>Player.MarkerHit</code></td>
</tr>
<tr>
<td>Player6MouseDown</td>
<td>Use <code>Player.MouseDown</code></td>
</tr>
<tr>
<td>Player6MouseMove</td>
<td>Use <code>Player.MouseMove</code></td>
</tr>
<tr>
<td>Player6MouseUp</td>
<td>Use <code>Player.MouseUp</code></td>
</tr>
</tbody>
</table>
Object Model Reference

The Windows Media Player Object Model Reference contains detailed information about the Windows Media Player ActiveX control object model. The information in this section is presented in a style designed for use with script languages like Microsoft JScript.

For information about how to use this reference with the C++ programming language, see Object Model Reference for C++.

For information about how to use this reference with the Microsoft Visual Basic programming language, see Using Windows Media Player with Visual Basic.

For information about how to use this reference with .NET Framework programming languages, see Embedding the Player Control in a .NET Framework Solution.

The Object Model Reference documents the following objects and their associated methods, properties, and events.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cdrom</td>
<td>Methods and properties for accessing a CD or DVD in its drive.</td>
</tr>
</tbody>
</table>
**CdromCollection**
Methods and properties for accessing a collection of CD-ROM drives or DVD-ROM drives.

**ClosedCaption**
Properties for including captions with a media clip.

**Controls**
Methods and properties representing the transport controls of Windows Media Player, such as Play, Stop, and Pause.

**DVD**
A property, methods, and events for working with DVDs.

**Error**
Methods and properties providing access to a collection of **ErrorItem** objects.

**ErrorItem**
Properties that provide information about errors.

**External**
Provides functionality to Web pages hosted in Windows Media Player.

**Media**
Methods and properties relating to multimedia clips.

**MediaCollection**
Methods that provide access to a collection of **Media** objects.

**MetadataPicture**
Properties for retrieving information on the image values of the **WM/Picture** metadata attribute.

**MetadataText**
Properties for retrieving metadata for complex textual metadata attributes.

**Network**
Properties relating to the network connection of Windows Media Player.

**Player**
Methods, properties, and events that Windows Media Player can be programmed to respond to.

**PlayerApplication**
Methods and properties for switching between a remoted Player control and the full mode of the Player. Can only be used with C++ programs that embed the control in remote mode.

**Playlist**
Methods and properties for manipulating lists of media clips.

**PlaylistArray**
A method and a property for accessing a collection of **Playlist** objects by index number.

**PlaylistCollection**
Methods for organizing a collection of **Playlist** objects.

**Settings**
Properties that allow the specification or retrieval of Windows Media Player settings.

**StringCollection**
A method and a property for manipulating collections of strings.

**See Also**
- Windows Media Player Object Model
Cdrom Object

TheCdromobject provides a way to access a CD or DVD in its drive.

TheCdromobject supports the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driveSpecifier</td>
<td>Retrieves the CD-ROM or DVD-ROM drive letter.</td>
</tr>
<tr>
<td>playlist</td>
<td>Retrieves a Playlist object representing the tracks on the CD currently in the CD-ROM drive or the root-level title entries for DVD.</td>
</tr>
</tbody>
</table>

TheCdromobject supports the following method:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eject</td>
<td>Ejects the CD or DVD from the drive.</td>
</tr>
</tbody>
</table>

TheCdromobject is accessed through the following method:

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>CdromCollection</td>
<td>item</td>
</tr>
</tbody>
</table>

For purposes of illustration, player.cdromCollection.item(index) is used in the reference syntax sections.

See Also

- Object Model Reference
The **driveSpecifier** property retrieves the CD-ROM or DVD-ROM drive letter.

**Syntax**

`player.cdromCollection.item(index).driveSpecifier`

**Possible Values**

This property is a read-only **String**.

**Remarks**

Typically, DVD-ROM drives can play CD media, but CD-ROM drives cannot play DVD media. This property retrieves a drive specifier for a zero-based drive index within the range retrieved using `CdromCollection.count`. The value retrieved takes the form `X:`, where `X` represents the drive letter.

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses `Cdrom.driveSpecifier` to fill an HTML text area named `myText` with a comma-separated list of available CD-ROM and DVD-ROM drives. The Player object was created with ID = "Player".

```javascript
// Allocate an array to store the drive specifiers.
var MYdriveSpecifiers = new Array();

// Loop through the available drives using `CdromCollection.count`.
for (var i = 0; i < Player.cdRomCollection.count; i++) {
    // For each available drive, add a corresponding item
    // to the MYdriveSpecifiers array.
    MYdriveSpecifiers[i] = Player.CDRomCollection.item(i).driveSpecifier;
}

// Write the array of drive letter specifiers to the text area.
myText.value = "Drive letters found: " + MYdriveSpecifiers;
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**Platform**: Windows XP or later for DVD.

**See Also**

- [Cdrom Object](#)
- [CdromCollection.count](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
Cdrom.eject

The **eject** method ejects the CD or DVD from the drive.

**Syntax**

```
player.cdromCollection.item(index).eject()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

If the drive door is open, this method closes the door.

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following example creates an HTML button element that uses **Cdrom.eject** to open the door of the drive that has drive specifier index zero. The player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON"
   NAME = "EJECTBUTTON"
   VALUE = "EJECT"
   OnClick = "Player.cdromCollection.item(0).eject()"
>
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.
Library: Use wmp.dll.

Platform: Windows XP or later for DVD.

See Also

- **Cdrom Object**
- **Player.playState**
- **Settings.mediaAccessRights**
- **Settings.requestMediaAccessRights**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Cdrom.playlist**

The `playlist` property retrieves a `Playlist` object representing the tracks on the CD currently in the CD-ROM drive or the root-level title entries for DVD.

**Syntax**

`player.cdromCollection.item(index).playlist`

**Possible Values**

This property is a read-only `Playlist` object.

**Remarks**

Typically, DVDs are organized into titles. Each title contains one or more chapters. Each DVD is authored differently, so how titles and chapters are used is up to the content author.

For DVD, this method retrieves a playlist that contains as the first item a `Media` object named "DVD". This object represents the DVD media. Playing the item results in the DVD playing from the beginning if it is the first play after inserting a new DVD, or resuming playback if the DVD is the same as the last DVD viewed. Setting this item as the current one during playback results in the DVD playing from the beginning.

Additional items (`Media` objects) in the playlist are DVD titles that are represented by nested playlists. When you specify `Controls.currentItem` to equal one of these nested playlist items, Windows Media Player automatically sets the nested playlist as `Player.currentPlaylist` after chapter playback begins. You can then use the `currentPlaylist` object properties, methods, and events to work with DVD chapters, which are also playlist items.
To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

### Example Code

The following example uses `Cdrom.playlist` to fill an HTML text element, named `myText`, with the titles of the audio CD currently in the first CD-ROM drive. Use `CdromCollection.count` to determine the number of available CD-ROM drives. The player object was created with ID = "Player".

```javascript
// Store the CD playlist object.
var pl = Player.cdromCollection.Item(0).Playlist;

// Iterate through the CD track list.
for(var i = 0; i < pl.count; i++){
    // Print each CD track name.
    myText.value += pl.Item(i).name + "\n";
}
```

### Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in `wmp.idl`; include `wmp.h`.

**Library:** Use `wmp.dll`.

**Platform:** Windows XP or later for DVD.

### See Also

- [Cdrom Object](#)
- `Settings.mediaAccessRights`
- `Settings.requestMediaAccessRights`

© 2000-2003 Microsoft Corporation. All rights reserved.

---

### CdromCollection Object

The **CdromCollection** object provides a way to organize and access a collection of CD-ROM drives or DVD-ROM drives.

The **CdromCollection** object supports the following property.
The CdromCollection object supports the following methods.

### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getByDriveSpecifier</td>
<td>Retrieves the Cdrom object associated with a particular drive letter.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves the Cdrom object at the given index.</td>
</tr>
</tbody>
</table>

The CdromCollection object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>cdromCollection</td>
</tr>
</tbody>
</table>

See Also

- Object Model Reference

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

## CdromCollection.count

The **count** property retrieves the number of available CD-ROM drives and DVD-ROM drives on the system.

### Syntax

`player.cdromCollection.count`

### Possible Values

This property is a read-only **Number (long)**.

### Remarks

To retrieve the value of this property, read access to **Media Library** is required. For more information, see Media Library Access.
Example Code

The following JScript example uses `CdromCollection.count` to display the number of CD-ROM drives and DVD-ROM drives available on the user's computer. The player object was created with ID = "Player".

```javascript
// Store the count of drives found on the computer.
var numCDROMS = Player.cdromCollection.count;

// Build the string to display to the user.
var displayString = numCDROMS + " drive(s) found."

// Show a message box with the count information.
alert(displayString);
```

Remarks

DVD-ROM drives are counted exactly like CD-ROM drives. However, the Windows Media Player control only supports DVD functionality for Windows XP or later operating systems. Typically, DVD-ROM drives can play CD media, but CD-ROM drives cannot play DVD media.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- [CdromCollection Object](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**CdromCollection.getByDriveSpecifier**

The `getByDriveSpecifier` method retrieves the `Cdrom` object associated with a particular drive letter.

**Syntax**

```javascript
player.cdromCollection.getByDriveSpecifier(driveSpecifier)
```
Parameters

driveSpecifier

String containing the drive letter followed by a colon (":" ) character.

Return Values

This method returns a Cdrom object.

Remarks

Drive specifiers must be given in the form X:, where X represents the drive letter.

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses CdromCollection.getByDriveSpecifier to retrieve the Cdrom object that corresponds to a drive letter provided by the user. An HTML text element was created, with NAME = "MyText", for user input. The player object was created with ID = "Player".

```javascript
// Store the drive letter provided by the user.
var DriveLetter = MyText.value;

// Append a colon to the drive letter to create a valid drive specifier.
DriveLetter += ":";

// Get the Cdrom object using the drive specifier.
var Drive = Player.cdRomCollection.getByDriveSpecifier(DriveLetter);

// Use the Cdrom object to open the drive door.
Drive.eject();
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Cdrom Object
- Cdrom.eject
- CdromCollection Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
**CdromCollection.item**

The `item` method retrieves the `Cdrom` object at the given index.

**Syntax**

```javascript
player.cdromCollection.item(index)
```

**Parameters**

- `index`  
  
  Number (`long`) containing the index.

**Return Values**

This method returns a `Cdrom` object.

**Remarks**

To use this method, read access to Media Library is required. For more information, see Media Library Access.

**Example Code**

The following JScript example uses `CdromCollection.item` to print the playlist name from each CD-ROM available on the computer. If the drive actually contains DVD content, Windows XP or later is required. An HTML TextArea element was created with ID = "playlists". The player object was created with ID = "Player".

```javascript
// Create an array to store the CD playlists.
var cdPlaylists = new Array();

// Loop through the available CD-ROM drives.
for (var i = 0; i < Player.cdromCollection.count; i++) {
  // Fill the cdPlaylists array with playlist objects.
  cdPlaylists[i] = Player.cdromCollection.item(i).Playlist;

  // Print each drive specifier.
  playlists.value += Player.cdromCollection.item(i).driveSpecifier + " ";

  // Print the name of each CD playlist to the text area.
  playlists.value += cdPlaylists[i].name + "\n";
}
```
ClosedCaption Object

The **ClosedCaption** object provides a way to include captions with a media clip. The captioning text is in a Synchronized Accessible Media Interchange (SAMI) file.

The **ClosedCaption** object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>captioningID</td>
<td>Specifies or retrieves the name of the frame or control displaying the captioning.</td>
</tr>
<tr>
<td>SAMIFilename</td>
<td>Specifies or retrieves the name of the file containing the information needed for closed captioning.</td>
</tr>
<tr>
<td>SAMILang</td>
<td>Specifies or retrieves the language displayed for closed captioning.</td>
</tr>
<tr>
<td>SAMILangCount</td>
<td>Retrieves the number of languages supported by the current SAMI file.</td>
</tr>
<tr>
<td>SAMIStyle</td>
<td>Specifies or retrieves the closed captioning style.</td>
</tr>
<tr>
<td>SAMIStyleCount</td>
<td>Retrieves the number of styles supported by the current SAMI file.</td>
</tr>
</tbody>
</table>

The **ClosedCaption** object supports the following methods.
The ClosedCaption object is accessed through the following property.

Object                  Property
Player                  closedCaption

See Also
- Adding Closed Captions to Digital Media
- Object Model Reference

### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSAMILangID</td>
<td>Retrieves the locale identifier (LCID) of a language supported by the current SAMI file.</td>
</tr>
<tr>
<td>getSAMILangName</td>
<td>Retrieves the name of a language supported by the current SAMI file.</td>
</tr>
<tr>
<td>getSAMISyleName</td>
<td>Retrieves the name of a style supported by the current SAMI file.</td>
</tr>
</tbody>
</table>

```
player.getCaption.langID
```

### ClosedCaption.captioningID

The captioningID property specifies or retrieves the name of the element displaying the captioning.

**Syntax**

```
player.closedCaption.captioningID
```

**Possible Values**

This property is a read/write String.

**Remarks**

The element name specified can be any HTML element in the Web page as long as it supports the innerHTML attribute. If the Web page contains multiple frames, the element name can only refer to an element in the same frame as the Player control.

**Example Code**
The following Microsoft JScript example uses `ClosedCaption.captioningID` to choose the area of a Web page used to display captions. Two HTML DIV elements were created, with ID = CC1 and ID = CC2, respectively. The player object was created with ID = "Player".

```xml
<!-- Create two HTML BUTTON elements to allow the user to choose a display region. -->
<INPUT TYPE = "BUTTON"  NAME = "SET1"  VALUE = "Move Caption to CC1"
    OnClick = "
    /* Clear the caption text from the other DIV */
    CC2.innerHTML = 'This is the CC2 DIV';
    /* Show the captions in the DIV named CC1. */
    Player.ClosedCaption.captioningID = 'CC1';
   ">

<INPUT TYPE = "BUTTON" NAME = "SET2" VALUE = "Move Caption to CC2"
    OnClick = "
    /* Clear the caption text from the other DIV */
    CC1.innerHTML = 'This is the CC1 DIV';
    /* Show the captions in the DIV named CC2. */
    Player.ClosedCaption.captioningID = 'CC2';
    >

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- [Adding Closed Captions to Digital Media](#)
- [ClosedCaption Object](#)

ClosedCaption.getSAMILangID

The `getSAMILangID` method retrieves the locale identifier (LCID) of a language supported by the current SAMI file.

Syntax
player.closedCaption.getSAMILangID(index)

Parameters

index

Number (long) specifying the index of the LCID to retrieve.

Return Values

This method returns a Number (long) containing the LCID of the language with the specified index.

Remarks

The languages in a SAMI file are indexed in the order shown in the file, starting with zero.

This method cannot be used until a digital media file is open (Player.openState is equal to 13).

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Adding Closed Captions to Digital Media
- ClosedCaption Object

ClosedCaption.getSAMILangName

The getSAMILangName method retrieves the name of a language supported by the current SAMI file.

Syntax

player.closedCaption.getSAMILangName(index)
Parameters

index

Number (long) specifying the index of the language name to retrieve.

Return Values

This method returns a String containing the name of the language as specified in the SAMI file.

Remarks

The languages in a SAMI file are indexed in the order shown in the file, starting with zero.

This method cannot be used until a digital media file is open (Player.openState is equal to 13).

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Adding Closed Captions to Digital Media
- ClosedCaption Object
- ClosedCaption.SAMILang

ClosedCaption.getSAMIStyleName

The getSAMIStyleName method retrieves the name of a style supported by the current SAMI file.

Syntax

player.closedCaption.getSAMIStyleName(index)

Parameters
Number (long) specifying the index of the style name to retrieve.

Return Values

This method returns a String containing the name of the style as specified in the SAMI file.

Remarks

The styles in a SAMI file are indexed in the order shown in the file, starting with zero.

This method cannot be used until a digital media file is open (Player.openState is equal to 13).

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Adding Closed Captions to Digital Media
- ClosedCaption Object
- ClosedCaption.SAMIStyle

ClosedCaption.SAMIFileName

The SAMIFileName property specifies or retrieves the name of the file containing the information needed for closed captioning.

Syntax

player.closedCaption.SAMIFilename

Possible Values
This property is a read/write String.

Remarks

The Synchronized Accessible Media Interchange (SAMI) file must use an .smi or .sami file name extension.

If you don't specify a value for SAMIFileName, this property retrieves a string containing the file name or URL associated with the current media item. This association can occur when a SAMI file is specified using the sami URL parameter, or automatically when the SAMI file has the same name as the digital media file name (except for the file name extension). If there is no SAMI file associated with the current media, this property retrieves an empty string (""").

Once you specify a value for SAMIFileName, that value persists until you specify a new value (or until a new media item is opened using the sami URL parameter). Therefore, you must specify a new value for this property before you play each new media item. That way, the new value for SAMIFileName will take effect when the next media item is opened (or when Player.close is called). Specifying a new value for SAMIFileName has no effect for the current media.

To cause Windows Media Player to return to using the SAMI file associated with a particular media item, specify a value for SAMIFileName equal to an empty string (""") before you play the next media item.

Example Code

The following three JScript examples use ClosedCaption.SAMIFileName to specify the name of a closed caption text file. The player object was created with ID = "Player".

```javascript
// Display the closed captions from a Web site.
Player.closedCaption.SAMIFileName="http://www.proseware.com/ccsample.smi";

// Display the closed captions from a local network.
// You must add an escape sequence of a backslash for every original backslash.
Player.closedCaption.SAMIFileName="\\yourservername\Public\ccsample.smi";

// Display the closed captions from a file on a local drive.
// Be sure to add the appropriate escape sequences.
Player.closedCaption.SAMIFileName="C:\WMSDK\WMPSDK9\samples\media\ccsample.smi";
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Adding Closed Captions to Digital Media
- ClosedCaption Object
- Player.close

Previous         Next
The `SAMILang` property specifies or retrieves the language displayed for closed captioning.

**Syntax**

`player.closedCaption.SAMILang`

**Possible Values**

This property is a read/write `String`.

**Remarks**

A SAMI file can contain text for one or many languages. The languages available for closed captioning are defined between the `<STYLE>` and `</STYLE>` tags in the SAMI file. A language identifier is specified with a unique alphanumeric string that is preceded by a period (.) The name specified for a language can be any string. For example, the following could be used to define US English:

```
.ENUUSC {Name:'English Captions' lang: en-US; SAMIType:CC;}
```

If no SAMI language is specified, the first language defined in the SAMI file is used by default.

The value you pass using `ClosedCaption.SAMILang` must match the `Name` attribute in the language specifier.

**Example Code**

The following JScript example uses `ClosedCaption.SAMILang` in an HTML SELECT element to specify the closed caption language. The player object was created with ID = "Player".

```html
<!-- Create the SELECT element. -->
<SELECT ID = 'CCLANG' NAME = 'CCLANG' LANGUAGE = 'JScript' onChange = "Player.closedCaption.SAMILang = CCLANG.value;">

/* Fill in the SELECT element options. */

/* Set the closed caption language when the SELECT element changes. */

/* Fill in the SELECT element options. */

<!-- Create the SELECT element. -->
```
ClosedCaption.SAMILangCount

The **SAMILangCount** property retrieves the number of languages supported by the current SAMI file.

**Syntax**

```csharp
player.closedCaption.SAMILangCount
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

This method cannot be used until a digital media file is open (**Player.openState** is equal to 13).

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- Adding Closed Captions to Digital Media
- ClosedCaption Object
ClosedCaption.SAMIStyle

The SAMIStyle property specifies or retrieves the closed captioning style.

Syntax

player.closedCaption.SAMIStyle

Possible Values

This property is a read/write String.

Remarks

A SAMI file can contain several format style definitions. SAMI styles are defined between the <STYLE> and </STYLE> tags in the SAMI file. A style is defined with a text string preceded by a # character. For example:

```
#Emphasis1 {Name: Big Bold Italic; font-size: 14pt; text-align: center;
  color: blue; font-family: sans-serif; font-weight: bold;
  font-style: italic;}
```

This specifies a style that produces a particular font.

If no SAMI style is specified, the first style defined in the SAMI file is used by default.

Example Code

The following JScript example creates an HTML SELECT element that uses closedCaption.SAMIStyle to change the appearance of the closed caption text. The player object was created with ID = "Player".

```
<!-- Create the SELECT element. -->
<SELECT ID = CCMode NAME = "CCMode" LANGUAGE = "JScript"

   /* Change the text style when the SELECT element changes. */
   onChange = "Player.closedCaption.SAMIStyle = CCMode.value;
   ">
```
/* Fill the SELECT list with options, set the default to Strong. */
<OPTION VALUE = "Normal">Normal
<OPTION VALUE = "Small">Small
<OPTION VALUE = "Big Bold Italic" SELECTED>Strong
</SELECT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Adding Closed Captions to Digital Media
- ClosedCaption Object

ClosedCaption.SAMIStyleCount

The SAMIStyleCount property retrieves the number of styles supported by the current SAMI file.

Syntax

player.closedCaption.SAMIStyleCount

Possible Values

This property is a read-only Number (long).

Remarks

This method cannot be used until a digital media file is open (Player.openState is equal to 13).

Requirements

Version: Windows Media Player 9 Series or later.
## Controls Object

The **Controls** object provides a way to manipulate the playback of media using the following properties and methods.

The **Controls** object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audioLanguageCount</code></td>
<td>Retrieves the number of supported audio languages.</td>
</tr>
<tr>
<td><code>currentAudioLanguage</code></td>
<td>Specifies or retrieves the locale identifier (LCID) of the audio language for playback</td>
</tr>
<tr>
<td><code>currentAudioLanguageIndex</code></td>
<td>Specifies or retrieves the index that corresponds to the audio language for playback.</td>
</tr>
<tr>
<td><code>currentItem</code></td>
<td>Specifies or retrieves the current media item.</td>
</tr>
<tr>
<td><code>currentMarker</code></td>
<td>Specifies or retrieves the current marker number.</td>
</tr>
<tr>
<td><code>currentPosition</code></td>
<td>Specifies or retrieves the current position in the media item in seconds from the beginning.</td>
</tr>
<tr>
<td><code>currentPositionString</code></td>
<td>Retrieves the current position in the media item as a <code>String</code>.</td>
</tr>
<tr>
<td><code>currentPositionTimecode</code></td>
<td>Specifies or retrieves the current position in the current media item using a time code format. This property currently supports SMPTE time code.</td>
</tr>
<tr>
<td><code>isAvailable</code></td>
<td>Retrieves whether a specified type of information is available or a</td>
</tr>
</tbody>
</table>
given action can be performed.

The **Controls** object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fastForward</strong></td>
<td>Starts fast play of the media item in the forward direction.</td>
</tr>
<tr>
<td><strong>fastReverse</strong></td>
<td>Starts fast play of the media item in the reverse direction.</td>
</tr>
<tr>
<td><strong>getAudioLanguageDescription</strong></td>
<td>Retrieves the description for the audio language corresponding to the specified index.</td>
</tr>
<tr>
<td><strong>getAudioLanguageID</strong></td>
<td>Retrieves the LCID for a specified audio language index.</td>
</tr>
<tr>
<td><strong>getLanguageName</strong></td>
<td>Retrieves the name of the audio language with the specified LCID.</td>
</tr>
<tr>
<td><strong>next</strong></td>
<td>Sets the current item to the next item in the playlist.</td>
</tr>
<tr>
<td><strong>pause</strong></td>
<td>Pauses the playing of the media item.</td>
</tr>
<tr>
<td><strong>play</strong></td>
<td>Causes the media item to start playing.</td>
</tr>
<tr>
<td><strong>playItem</strong></td>
<td>Causes the current media item to start playing, or resumes play of a paused item.</td>
</tr>
<tr>
<td><strong>previous</strong></td>
<td>Sets the current item to the previous item in the playlist.</td>
</tr>
<tr>
<td><strong>step</strong></td>
<td>Causes the current video media item to freeze playback on the next frame.</td>
</tr>
<tr>
<td><strong>stop</strong></td>
<td>Stops the playing of the media item.</td>
</tr>
</tbody>
</table>

The **Controls** object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>controls</td>
</tr>
</tbody>
</table>

See Also

- [Object Model Reference](#)
Controls.audioLanguageCount

The **audioLanguageCount** property retrieves the count of supported audio languages.

**Syntax**

`player.controls.audioLanguageCount`

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

For Windows Media-based content, properties and methods related to language selection only support a single output.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Controls Object](#)
- [Controls.currentAudioLanguage](#)
- [Controls.currentAudioLanguageIndex](#)
- [Controls.getAudioLanguageDescription](#)
- [Controls.getAudioLanguageID](#)
- [Controls.getLanguageName](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Syntax

player.controls.currentAudioLanguage

Possible Values

This property is a read/write Number (long).

Remarks

An LCID uniquely identifies a particular language dialect, called a locale.

For Windows Media-based content, properties and methods related to language selection only support a single output.

When working with DVD content, specifying an LCID will cause the first available audio track having the specified language ID to be selected.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Controls.audioLanguageCount
- Controls.currentAudioLanguageIndex
- Controls.getAudioLanguageDescription
- Controls.getAudioLanguageID
- Controls.getLanguageName
Syntax

player.controls.currentAudioLanguageIndex

Possible Values

This property is a read/write Number (long).

Remarks

For Windows Media-based content, properties and methods related to language selection only support a single output.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Controls.audioLanguageCount
- Controls.currentAudioLanguage
- Controls.getAudioLanguageDescription
- Controls.getAudioLanguageID
- Controls.getLanguageName

© 2000-2003 Microsoft Corporation. All rights reserved.
This property is a read/write **Media** object.

**Remarks**

This method works only with items in *Player.currentPlaylist*. Calling **currentItem** with a reference to a saved media item is not supported.

**Example Code**

The following JScript example uses **currentItem** to set the player control current media item to the selected value in an HTML SELECT element. The current playlist was first specified by using *PlaylistCollection.getByName*. The player object was created with ID = "Player".

```html
<SELECT ID = playItem NAME = "playItem" LANGUAGE = "JScript"
  onChange = "Player.controls.currentItem = Player.currentPlaylist.item(playItem.value)"
>
  /* Specify the current item when the selected list value changes. */
  /* Fill the SELECT element list with three items that match
  the songs in the playlist. */
  <OPTION VALUE = 0 >Laure
  <OPTION VALUE = 1 >Jeanne
  <OPTION VALUE = 2 >House
</SELECT>
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Controls Object**
- **Media Object**
- **PlaylistCollection.getByName**

© 2000-2003 Microsoft Corporation. All rights reserved.
The **currentMarker** property specifies or retrieves the current marker number.

**Syntax**

`player.controls.currentMarker`

**Possible Values**

This property is a read/write **Number (long)** with a default value of zero.

**Remarks**

Setting **currentMarker** causes playback to start from the specified marker. Before attempting to set **currentMarker**, determine whether a file has markers and how many it has by using **markerCount**. If a file has no markers, setting **currentMarker** to anything but zero results in an error. Setting **currentMarker** to a number higher than **markerCount** also results in an error.

The **currentMarker** property always returns the current or last marker, which means the actual file position can be either at the current marker or before the next marker. Markers are numbered beginning at 1, so if a file has markers, you can set **currentMarker** to zero to change the file position to zero.

Until the current media item is set (using **Player.URL** or **Player.currentMedia**), **currentMarker** returns zero.

**Example Code**

The following JScript example uses **currentMarker** to start video playback from the marker that corresponds to the **selectedIndex** property of an HTML SELECT element. The player object was created with ID = "Player".

```xml
<SELECT ID = "markers" NAME = "markers" LANGUAGE = "JScript"
    /* Seek to the marker number that corresponds to the SELECT element
     * selectedIndex value when the list selection changes. */
    onchange = "Player.controls.currentMarker = markers.selectedIndex + 1;
    ">
    /* Fill the SELECT element with the marker identifiers. */
    <OPTION SELECTED>Sunrise
    <OPTION>Car chase
    <OPTION>Happy ending
</SELECT>
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [Controls Object](#)
- [Media.markerCount](#)
The `currentPosition` property specifies or retrieves the current position in the media item in seconds from the beginning.

**Syntax**

```javascript
player.controls.currentPosition
```

**Possible Values**

This property is a read/write `Number (double)`.

**Example Code**

The following example uses `currentPosition` to seek to a position provided by the user. An HTML BUTTON element is created to execute the JScript code. An HTML TEXT input element, named `setPosition`, was created to accept a value, in seconds, from the user. The player object was created with ID = "Player".

```html
<input type = "button"  id = "Set"  name = "Set"  value = "Set Position"
/* Check to be sure the TEXT element contains a valid value. */
if (!isNaN(setPosition.value) && (setPosition.value != ''))
/* Set the current position when the user clicks the button. */
onClick = "Player.controls.currentPosition = setPosition.value;
">"
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**
The `currentPositionString` property retrieves the current position in the media item as a `String` formatted as HH:MM:SS (hours, minutes, and seconds).

**Syntax**

```javascript
player.controls.currentPositionString
```

**Possible Values**

This property is a read-only `String`.

**Remarks**

If the media item is less than an hour long, the HH: portion is not included.

**Example Code**

The following JScript example starts an HTML timer that displays the current position of the media file at one-second intervals. An HTML TEXT element named MyText was created to display the current position. The player object was created with ID = "Player".

```javascript
var timer = window.setInterval("MyText.value = Player.controls.currentPositionString",1000)
```

**Note** You should include code to stop the timer when the media is stopped or paused.

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**
Controls.currentPositionTimecode

The `currentPositionTimecode` property specifies or retrieves the current position in the current media item using a time code format. This property currently supports SMPTE time code.

**Syntax**

```javascript
player.controls.currentPositionTimecode
```

**Possible Values**

This property is a read/write `String`.

**Remarks**

SMPTE time code provides a standard way of identifying an individual video frame, which is useful for synchronizing playback. If a digital media file supports SMPTE time code, Windows Media Player can retrieve the current time code position information or seek to a video frame identified by a particular time code `String`.

SMPTE time code identifies a particular frame by the number of hours, minutes, seconds, and frames that separate it from a particular reference frame—the frame designated as time zero. Usually the time zero frame is the start of the file and a particular SMPTE time code value represents the elapsed time since the start of the file.

The time code `String` is in the format `[range]hh:mm:ss.ff` where `[range]` represents the range, `hh` represents hours, `mm` represents minutes, `ss` represents seconds, and `ff` represents frames. When specifying a value using `currentPositionTimecode`, you must include all eight digits using zeroes as placeholders.

The `[range]` specifier corresponds to the `wRange` member of the Windows Media Format `WMT_TIMECODE_EXTENSION_DATA` structure. For more information about time code ranges, see the Windows Media Format SDK.

Specifying and retrieving `currentPositionTimecode` is only supported for files that contain SMPTE time code information.

**Example Code**

The following code example specifies `currentPositionTimecode` as 1 hour, zero minutes, 30 seconds, and 5
// Seek to a frame using SMPTE time code.
Player.controls.currentPositionTimecode = "[00000]01:00:30.05";

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- **Controls Object**

Controls.fastForward

The `fastForward` method starts fast play of the media item in the forward direction.

Syntax

`player.controls.fastForward()`

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

The `fastForward` method plays the clip back at five times the normal speed. Invoking `fastForward` changes the `Settings.rate` property to 5.0. If `rate` is subsequently changed, or if `play` or `stop` is called, Windows Media Player will cease fast forwarding.

The `fastForward` method does not work for live broadcasts and certain media types. To determine whether you
can fast forward in a clip, call `isAvailable("FastForward")`.

**Example Code**

The following example creates an HTML BUTTON element that uses `fastForward` to start fast play of the media item. The player object was created with ID = "Player".

```html
<input type="button" id="FF" name="FF" value=">>" /
  /* Execute JScript when the BUTTON is clicked.
   Check first to make sure fast-forward mode is available
   for this particular media item */
  onClick = "if (Player.controls.isAvailable('FastForward'))
                 Player.controls.fastForward();
               ">
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- Controls Object
- Controls.isAvailable
- Controls.play
- Controls.stop
- Settings.rate

© 2000-2003 Microsoft Corporation. All rights reserved.

**Controls.fastReverse**

The `fastReverse` method starts fast scanning of the media item in the reverse direction.

**Syntax**

```javascript
player.controls.fastReverse()
```

**Parameters**
This method takes no parameters.

Return Values

This method does not return a value.

Remarks

The `fastReverse` method scans the clip in reverse at five times the normal speed, displaying only the key frames if it is a video file. Invoking `fastReverse` changes the `Settings.rate` property to –5.0. If `rate` is subsequently changed, or if `play` or `stop` is called, Windows Media Player will cease fast reverse.

If the item is part of a playlist, `fastReverse` stops at the beginning of the current track. For instance, if track 3 is in `fastReverse`, when the beginning of track 3 is reached, Windows Media Player will not go to track 2. The `Settings.playCount` property is ignored for `fastReverse`.

If you call `fastForward` while `fastReverse` is in effect, `fastReverse` will be stopped and `fastForward` will begin.

This method does not work for live broadcasts and certain media types. To determine whether you can use fast reverse in a clip, call `isAvailable("FastReverse")`.

Example Code

The following example creates an HTML BUTTON element that uses `fastReverse` to start fast-reverse play of the media item. The player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON"  ID = "REW"  NAME = "REW"  VALUE = "<<"
        /* Execute JScript when the BUTTON is clicked.
           Check first to make sure fast-reverse mode is available
           for this particular media item */
        onClick = "if (Player.controls.isAvailable('FastReverse'))
                   Player.controls.fastReverse();
                   ">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Controls.fastForward
- Controls.isAvailable
- Controls.play
- Controls.stop
- Settings.rate
The `getAudioLanguageDescription` method retrieves the description for the audio language corresponding to the specified index.

### Syntax

```javascript
player.controls.getAudioLanguageDescription(index)
```

### Parameters

`index`

Number (long) specifying the audio language index.

### Return Values

This method returns a `String` value.

### Remarks

For Windows Media-based content, properties and methods related to language selection only support a single output.

### Requirements

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in `wmp.idl`; include `wmp.h`.

**Library**: Use `wmp.dll`.

### See Also

- [Controls Object](#)
- [Controls.audioLanguageCount](#)
- [Controls.currentAudioLanguage](#)
- [Controls.currentAudioLanguageIndex](#)
- [Controls.getAudioLanguageID](#)
The `getAudioLanguageID` method retrieves the locale identifier (LCID) for a specified audio language index.

**Syntax**

```plaintext
player.controls.getAudioLanguageID(index)
```

**Parameters**

- `index`
  
  `Number (long)` specifying the audio language index.

**Return Values**

This method returns a `Number (long)`.

**Remarks**

An LCID uniquely identifies a particular language dialect, called a locale.

For Windows Media-based content, properties and methods related to language selection only support a single output.

**Requirements**

- **Version**: Windows Media Player 9 Series or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- `Controls Object`
- `Controls.audioLanguageCount`
The `getLanguageName` method retrieves the name of the audio language with the specified locale identifier (LCID).

**Syntax**

```csharp
player.controls.getLanguageName(LCID)
```

**Parameters**

`LCID`  
Number (`long`) specifying the LCID.

**Return Values**

This method returns a `String`.

**Remarks**

An LCID uniquely identifies a particular language dialect, called a locale.

For Windows Media-based content, properties and methods related to language selection only support a single output.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in `wmp.idl`; include `wmp.h`.

**Library:** Use `wmpl.dll`.
The `isAvailable` property indicates whether a specified type of information is available or a specified action can be performed.

**Syntax**

```javascript
player.controls.isAvailable(name)
```

**Parameters**

*name*

**String** containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>currentItem</code></td>
<td>Determines whether the user can set the <code>currentItem</code> property.</td>
</tr>
<tr>
<td><code>currentMarker</code></td>
<td>Determines whether the user can seek to a specific marker.</td>
</tr>
<tr>
<td><code>currentPosition</code></td>
<td>Determines whether the user can seek to a specific position in the file. Some files do not support seeking.</td>
</tr>
<tr>
<td><code>fastForward</code></td>
<td>Determines whether the file supports fast forwarding and whether that functionality can be invoked. Many file types (or live streams) do not support FastForward.</td>
</tr>
<tr>
<td><code>fastReverse</code></td>
<td>Determines whether the file supports FastReverse and whether that functionality can be invoked. Many file types (or live streams) do not support FastReverse.</td>
</tr>
</tbody>
</table>
next Determines whether the user can seek to the next entry in a playlist.

pause Determines whether the pause method is available.

play Determines whether the play method is available.

previous Determines whether the user can seek to the previous entry in a playlist.

step Determines whether the step method is available during playback.

stop Determines whether the stop method is available.

Return Values

This method returns a Boolean value.

Example Code

The following example creates an HTML BUTTON element that seeks to the starting position of the current media item. The JScript code uses isAvailable to verify that the file supports the seek operation. The player object was created with ID = "Player".

```html
<input type = "BUTTON"  id = "START"  name = "START"  value = "Seek To Zero"

/* If supported, seek the media file to position zero. */
onClick = "if (Player.controls.isAvailable('CurrentPosition'))
    Player.controls.currentPosition = 0;
    " >
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
Controls.next

The `next` method sets the current item to the next item in the playlist.

Syntax

`player.controls.next()`

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

If the playlist is on the last entry when `next` is invoked, the first entry in the playlist will become the current one.

For server-side playlists, this method skips to the next item in the server-side playlist, not the client playlist.

When playing a DVD, this method skips to the next logical chapter in the playback sequence, which may not be the next chapter in the playlist. When playing DVD stills, this method skips to the next still.

Example Code

The following example creates an HTML BUTTON element that uses `next` to move to the next item in the current playlist. The player object was created with ID = "Player".

```html
<input type = "BUTTON"  id = "NEXT"  name = "NEXT"  value = ">>|" 
onclick = "
/* Check first to be sure the operation is valid. */
if (Player.controls.isAvailable('Next'))

/* Move to the next item in the playlist. */
Player.controls.next();
">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- [Controls Object](#)
- [Controls.previous](#)
Controls.pause

The `pause` method pauses the playing of the media item.

Syntax

```javascript
player.controls.pause()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

When a file is paused, Windows Media Player does not give up any system resources, such as the audio device.

Certain media types cannot be paused, such as live streams. To determine whether a particular media type can be paused, use `Controls.isAvailable('Pause')`.

Example Code

The following example creates an HTML BUTTON element that uses `pause` to pause playback. The player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON" ID = "PAUSE" NAME = "PAUSE" VALUE = "||"
       onClick = "
           /* Check first to be sure the operation is valid. */
           if (Player.controls.isAvailable('Pause'))
               /* Pause the player. */
               Player.controls.pause();
       ">
```

Requirements
Controls.play

The play method causes the current media item to start playing, or resumes play of a paused item.

Syntax

player.controls.play()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

If this method is called while fast-forwarding or rewinding, the value of Settings.rate is set to 1.0.

Example Code

The following example creates an HTML BUTTON element that uses play to play the current media item. The player object was created with ID = "Player".

```html
<input type = "BUTTON"  ID = "PLAY"  NAME = "PLAY"  VALUE = "Play"
       onclick = "/* Check first to be sure the operation is valid. */"
```
if (Player.controls.isAvailable('Play'))

    /* Start playback. */
    Player.controls.play();

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object

Controls.playItem

The playItem method plays the specified media item.

Syntax

player.controls.playItem(theMediaItem)

Parameters

theMediaItem

Media object to be played.

Return Values

This method does not return a value.

Remarks

The media item will be loaded and played automatically, regardless of the value of the Settings.autoStart property. To load an item without playing it automatically, set Settings.autoStart to false and assign a value to
Player.URL, after which **play** can be called to start playing the item.

**Note**  **playItem** works only with items in **currentPlaylist**. Calling **playItem** with a reference to a saved media item is not supported.

**Example Code**

The following JScript example uses **playItem** to play a media item from the current playlist. The item to play is chosen based upon its position in the playlist. The player object was created with ID = "Player".

```javascript
// Declare a variable to hold the position of the media item
// in the current playlist. An arbitrary value is supplied here.
var index = 3

// Retrieve the media item at the fourth position in the current playlist.
var media = Player.currentPlaylist.item(index);

// Play the media item.
Player.controls.playItem(media);
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Controls Object](#)
- [Playlist.item](#)

---

**Controls.previous**

The **previous** method sets the current item to the previous item in the playlist.

**Syntax**

```javascript
player.controls.previous()
```
Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

If the playlist is on the first entry when `previous` is invoked, the last entry in the playlist will become the current one.

Example Code

The following example creates an HTML BUTTON element that uses `previous` to move to the preceding item in the current playlist. The player object was created with ID = "Player".

```html
<input type = "button" id = "PREV" name = "PREV" value = "|<<"
     onclick = "
       /* Check first to be sure the operation is valid. */
       if (Player.controls.isAvailable('Previous'))
         /* Move to the preceding item in the playlist. */
         Player.controls.previous();
     ">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Controls.next
- Controls.stop
Controls.step

The `step` method causes the current video media item to freeze playback on the next frame or the previous frame.

Syntax

```
player.controls.step(frameCount)
```

Parameters

`frameCount`

Number (`long`) indicating how many frames to step before freezing. Must currently be set to 1 or -1.

Return Values

This method does not return a value.

Remarks

This method currently only supports the parameters 1 or -1, so you can only step one frame at a time.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- DVD Object

© 2000-2003 Microsoft Corporation. All rights reserved.

Controls.stop
The stop method stops the playing of the media item.

Syntax

player.controls.stop()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

This method causes Windows Media Player to release any system resources it is using, such as the audio device. The current media file, however, is not released.

When the player is stopped, the track rewinds to the beginning. Calling play will then begin playback of the clip from the beginning. To halt a play operation without changing the current position, use the pause method.

Example Code

The following example creates an HTML BUTTON element that uses stop to stop playback. The player object was created with ID = "Player".

```html
<input type = "button" id = "STOP" name = "STOP" value = "Stop"
onClick = "
    /* Check first to be sure the operation is valid. */
    if (Player.controls.isAvailable('Stop'))

    /* Stop the player. */
    Player.controls.stop();
    ”>
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Controls.next
- Controls.pause
- Controls.previous
The **DVD** object provides properties and methods for working with DVDs.

The **DVD** object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>Retrieves the current domain of the DVD.</td>
</tr>
<tr>
<td>isAvailable</td>
<td>Retrieves whether a specified type of information is available or a given action can be performed.</td>
</tr>
</tbody>
</table>

The **DVD** object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>Changes the display from a submenu to its parent menu.</td>
</tr>
<tr>
<td>resume</td>
<td>Changes to playback mode from menu mode, resuming at the same position as when the menu was invoked.</td>
</tr>
<tr>
<td>titleMenu</td>
<td>Stops playback and displays the title menu.</td>
</tr>
<tr>
<td>topMenu</td>
<td>Stops playback and displays the root menu.</td>
</tr>
</tbody>
</table>

The **DVD** object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>dvd</td>
</tr>
</tbody>
</table>

See Also

- [Object Model Reference](#)
DVD.back

The `back` method returns the display from a submenu to its parent menu.

**Syntax**

```cpp
player.dvd.back()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

Every DVD is authored differently. Some DVDs are authored so that the `back` method is available from the root menu, but when invoked, it will do nothing.

**Requirements**

**Version:** Windows Media Player for Windows XP or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**Platform:** Windows XP or later.

**See Also**

- [DVD Object](#)
The **domain** property retrieves the current domain of the DVD.

**Syntax**

```plaintext
player.dvd.domain
```

**Possible Values**

This property is a read-only **String** containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstPlay</td>
<td>Performing default initialization of a DVD disc.</td>
</tr>
<tr>
<td>videoManagerMenu</td>
<td>Displaying menus for the entire disc. Also known as <code>topMenu</code> for Windows Media Player. Commonly called the title menu or the top menu.</td>
</tr>
<tr>
<td>videoTitleSetMenu</td>
<td>Displaying menus for current title set. Also known as <code>titleMenu</code> for Windows Media Player. Commonly called the root menu.</td>
</tr>
<tr>
<td>title</td>
<td>Usually displaying the current title.</td>
</tr>
<tr>
<td>stop</td>
<td>The DVD Navigator is in the DVD Stop domain.</td>
</tr>
<tr>
<td>undefined</td>
<td>Player is not in any DVD domain.</td>
</tr>
</tbody>
</table>

**Remarks**

Every DVD is authored differently. Some DVDs do not contain the firstPlay, videoManagerMenu, or videoTitleSetMenu domains.

**Requirements**

- **Version:** Windows Media Player for Windows XP or later.
- **Header:** Defined in wmp.idl; include wmp.h.
- **Library:** Use wmp.dll.
- **Platform:** Windows XP or later.

**See Also**

- [DVD Object](#)
The `isAvailable` property indicates whether a specified type of information is available or a specified action can be performed.

**Syntax**

```plaintext
player.dvd.isAvailable(name)
```

**Parameters**

`name`

String containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>Determines whether the <code>back</code> method is available.</td>
</tr>
<tr>
<td>dvd</td>
<td>Determines whether the DVD is loaded.</td>
</tr>
<tr>
<td>dvdDecoder</td>
<td>Determines whether the DVD decoder is installed on system.</td>
</tr>
<tr>
<td>resume</td>
<td>Determines whether the <code>resume</code> method is available.</td>
</tr>
<tr>
<td>titleMenu</td>
<td>Determines whether the <code>titleMenu</code> method is available.</td>
</tr>
<tr>
<td>topMenu</td>
<td>Determines whether the <code>topMenu</code> method is available. Commonly called the root menu.</td>
</tr>
</tbody>
</table>

**Return Values**

This method returns a `Boolean` value indicating whether the specified parameter is available.

**Remarks**

The DVD features of Windows Media Player will not work on computers that do not have third-party DVD decoders installed. You can determine whether a decoder is available by calling `isAvailable("dvdDecoder")`.

Every DVD is authored differently. The methods available during DVD playback and navigation depend on
how the DVD is authored.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

Platform: Windows XP or later.

See Also

- DVD Object

DVD.resume

The resume method returns to playback mode from menu mode at the same title position as when the menu was invoked.

Syntax

player.dvd.resume()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.
Library: Use wmp.dll.

Platform: Windows XP or later.

See Also

- DVD Object

DVD.titleMenu

The `titleMenu` method stops title playback and displays the title menu.

Syntax

```c
player.dvd.titleMenu()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

Every DVD is authored differently. The DVD must contain a menu for this method to work. Some DVDs are authored so that the `topMenu` and `titleMenu` methods open the same menu. The `titleMenu` method usually invokes the title menu but it may invoke the top menu instead, if there is no title menu available.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

Platform: Windows XP or later.
The `topMenu` method stops title playback and displays the top (or root) menu for the current title.

**Syntax**

```plaintext
player.dvd.topMenu()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

Every DVD is authored differently. The DVD must contain a menu for this method to work. Some DVDs are authored so that the `topMenu` and `titleMenu` methods open the same menu. The `topMenu` method usually invokes the top (or root) menu but it may invoke the title menu instead, if there is no root menu available.

**Requirements**

**Version:** Windows Media Player for Windows XP or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**Platform:** Windows XP or later.
Error Object

The Error object provides access to a collection of ErrorItem objects.

The Error object supports the following property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorCount</td>
<td>Retrieves the number of errors in the error queue.</td>
</tr>
</tbody>
</table>

The Error object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearErrorQueue</td>
<td>Clears the errors from the error queue.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves an ErrorItem object from the error queue.</td>
</tr>
<tr>
<td>webHelp</td>
<td>Launches the Microsoft Windows Media Player Web Help page to display further information about the error.</td>
</tr>
</tbody>
</table>

The Error object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object Property</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>error</td>
</tr>
</tbody>
</table>

See Also

- Object Model Reference
Error.clearErrorQueue

The clearErrorQueue method clears the errors from the error queue.

Syntax

player.error.clearErrorQueue()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

This method is useful to clear the error queue after a series of errors has been processed.

Example Code

The following JScript example uses Error.clearErrorQueue in an event handler to empty the error queue after all error descriptions are displayed. The player object was created with ID = "Player".

<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = error()>

// Store the number of errors in the queue.
var max = Player.error.errorCount

// Loop through the list of errors.
for (var i = 0; i < max; i++){
   // Get the error description for this item.
   var errDesc = Player.error.item(i).errorDescription;

   // Display the error message.
   alert(errDesc);
}

// Clear the error queue to prepare for the next group of errors.
Player.error.clearErrorQueue();

</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.
Error.errorCount

The `errorCount` property retrieves the number of errors in the error queue.

Syntax

```
player.error.errorCount
```

Possible Values

This property is a read-only `Number (long)`.

Example Code

The following JScript example uses `Error.errorCount` in an event handler to alert the user about the most recent error in the error queue. The player object was created with ID = "Player".

```
<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = error()>

// Store the number of errors in the queue.
var max = Player.error.errorCount;

// Get the description of the last error. Error items start at zero,
// so the item index will always be one less than the error count.
var errDesc = Player.error.item(max-1).errorDescription;

// Display the error description.
alert(errDesc);

</SCRIPT>
```

Requirements

Version: Windows Media Player version 7.0 or later.
Error.item

The item method retrieves an ErrorItem object from the error queue.

Syntax

player.error.item(index)

Parameters

index

Number (long) containing the index of the ErrorItem object to be retrieved.

Return Values

This method returns an ErrorItem object.

Remarks

Windows Media Player can generate a number of errors in response to an error condition. This method allows the retrieval of a specific error in the queue using an index number. The index numbers for the error queue begin with zero.

Example Code

The following JScript example uses the Error.item object in an event handler to alert the user to the most recent error. The player object was created with ID = "Player".

<SCRIPT LANGUAGE="JScript" FOR=Player EVENT=error()>

// Store the most recent error item number.
var max = Player.error.errorCount - 1

// Store the most recent error in an error item object.
var errItem = Player.error.item(max);

// Use the error item object to store the error info.
errDesc = errItem.errorDescription;
errNum = errItem.errorCode;

// Display the error info.
alert(errNum + "\n" + errDesc);

</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Error Object
- ErrorItem Object

Error.webHelp

The webHelp method launches the Microsoft Windows Media Player Web Help page to display further information about the first error in the error queue (index zero).

Syntax

player.error.webHelp()

Parameters

This method takes no parameters.

Return Values
This method does not return a value.

Remarks

The Web Help pages always contain the latest and most detailed information about Windows Media Player errors. This method automatically transfers the other information needed by Web Help, such as the operating system version being used.

To access the Web Help pages directly, use the error code and support center links provided in the See Also list.

Example Code

The following example creates an HTML BUTTON element that launches the Microsoft Windows Media Player Web Help page. The player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON" NAME = "WHBUTTON" VALUE = "More Info"
   OnClick = "Player.error.webHelp();">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Error Object
- Windows Media Player Error Code Information
- Windows Media Player Support Center

© 2000-2003 Microsoft Corporation. All rights reserved.

---

ErrorItem Object

The ErrorItem object provides a way to access error information.

The ErrorItem object supports the following properties.
The **ErrorItem** object is accessed through the following methods.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>condition</strong></td>
<td>Retrieves a value indicating the condition for the error.</td>
</tr>
<tr>
<td><strong>customUrl</strong></td>
<td>Retrieves the URL of a Web site that displays specific information about codec download failure.</td>
</tr>
<tr>
<td><strong>errorCode</strong></td>
<td>Retrieves the current error code.</td>
</tr>
<tr>
<td><strong>errorContext</strong></td>
<td>Retrieves a value indicating the context of the error.</td>
</tr>
<tr>
<td><strong>errorDescription</strong></td>
<td>Retrieves a description of the error.</td>
</tr>
<tr>
<td><strong>remedy</strong></td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

For purposes of illustration, `player.error.item(index)` is used in the reference syntax sections.

See Also

- [Object Model Reference](#)

---

**Property Method**

**Object**

**Error**

item

**Media**

error

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**ErrorItem.condition**

The **condition** property retrieves a value indicating the condition for the error.

**Syntax**

`player.error.item(index).condition`

**Possible Values**

This property is a read-only **Number (long)** that represents the condition code.
Remarks

The condition code is a value that is used by Microsoft to provide additional information for technical support personnel.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- ErrorItem Object

ErrorItem.customUrl

The customURL property retrieves the URL of a Web site that displays specific information about codec download failure.

Syntax

player.error.item(index).customURL

Possible Values

This property is a read-only String.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also
ErrorItem.errorCode

The **errorCode** property retrieves the current error code.

**Syntax**

```javascript
player.error.item(index).errorCode
```

**Possible Values**

This property is a read-only **Number (long)**.

**Example Code**

The following JScript example uses **ErrorItem.errorCode** in an event handler to display the error code to the user. The player object was created with ID = "Player".

```javascript
<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = error()>
// Get the error code for the first error item.
errNum = Player.error.item(0).errorCode;

// Display the error information.
var message = "Error number: " + errNum);
message += "<BR>");
message += "Use your BACK button to return ");
message += "to the previous page.");
document.write(message);

</SCRIPT>
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.
ErrorItem Object

ErrorItem.errorDescription

ErrorItem.errorContext

The errorContext property retrieves a value indicating the context of the error.

Syntax

player.error.item(index).errorContext

Possible Values

This property is a read-only String that represents the error context code.

Remarks

The error context is information that is used by Microsoft to provide additional information for technical support personnel.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- ErrorItem Object

© 2000-2003 Microsoft Corporation. All rights reserved.
ErrorItem.errorDescription

The errorDescription property retrieves a description of the error.

Syntax

player.error.item(index).errorDescription

Possible Values

This property is a read-only String.

Example Code

The following JScript example uses ErrorItem.errorDescription in an event handler to display the error message to the user. The player object was created with ID = "Player".

<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = error()>

// Get the error description for the first error item.
errDesc = Player.error.item(0).errorDescription;

// Display the error code.
var message = "Error: " + errDesc;
message += "<BR>";
message += "Use your BACK button to return ";
message += "to the previous page.";
document.write(message);

</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- ErrorItem Object
The External object can provide functionality to Web pages hosted in Windows Media Player that were opened using `IWMPPlayerServices::setTaskPaneURL`.

Note The properties, methods, and events exposed by the External object are available for use with Windows Media Player 9 Series in the Windows XP operating system. They may be altered or unavailable in subsequent versions.

The External object exposes the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>appColorLight</code></td>
<td>Retrieves the current color of the Windows Media Player user interface.</td>
</tr>
<tr>
<td><code>version</code></td>
<td>Retrieves the current version of Windows Media Player.</td>
</tr>
</tbody>
</table>

The External object exposes the following event.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OnColorChange</code></td>
<td>Occurs when the color of the Windows Media Player user interface changes.</td>
</tr>
</tbody>
</table>

See Also

- [IWMPPlayerServices::setTaskPaneURL](#)
- [Object Model Reference](#)
- [Remoting the Windows Media Player Control](#)
**External.appColorLight**

The `appColorLight` property retrieves the current color of the Windows Media Player user interface.

**Syntax**

```javascript
window.external.appColorLight
```

**Possible Values**

This property is a read-only `String`.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [External Object](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**External.OnColorChange Event**

The `OnColorChange` event occurs when the color of the Windows Media Player user interface changes.

**Syntax**

```javascript
window.external.OnColorChange = functionname
```

**Parameters**

`functionname`

The name of the function in script that Windows Media Player calls when the event occurs

**Requirements**

Windows Media Player 9 Series or later.
External.version

The `version` property retrieves the current version of Windows Media Player.

Syntax

```javascript
window.external.version
```

Possible Values

This property is a read-only `String`.

Requirements

Windows Media Player 9 Series or later.

See Also

- **External Object**

Media Object

The `Media` object provides a way to specify or retrieve properties of a multimedia clip, using the following
The Media object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeCount</td>
<td>Retrieves the number of attributes that can be queried and/or set for the media item.</td>
</tr>
<tr>
<td>duration</td>
<td>Retrieves the duration in seconds of the current media item.</td>
</tr>
<tr>
<td>durationString</td>
<td>Retrieves a String value indicating the duration of the current media item in HH:MM:SS format.</td>
</tr>
<tr>
<td>error</td>
<td>Retrieves an ErrorItem object if the media item has an error condition.</td>
</tr>
<tr>
<td>imageSourceHeight</td>
<td>Retrieves the height of the current media item in pixels.</td>
</tr>
<tr>
<td>imageSourceWidth</td>
<td>Retrieves the width of the current media item in pixels.</td>
</tr>
<tr>
<td>markerCount</td>
<td>Retrieves the number of markers in the media item.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies or retrieves the name of the media item.</td>
</tr>
<tr>
<td>sourceURL</td>
<td>Retrieves the URL of the media item.</td>
</tr>
</tbody>
</table>

The Media object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttributeCountByType</td>
<td>Retrieves the number of attributes associated with the specified attribute name and language.</td>
</tr>
<tr>
<td>getAttributeName</td>
<td>Retrieves the name of the attribute corresponding to the specified index.</td>
</tr>
<tr>
<td>getItemInfo</td>
<td>Retrieves the value of the specified attribute for the media item.</td>
</tr>
<tr>
<td>getItemInfoByAtom</td>
<td>Retrieves the value of the attribute with the specified index number.</td>
</tr>
<tr>
<td>getItemInfoByType</td>
<td>Retrieves the value of the attribute corresponding to the specified attribute name, language, and index.</td>
</tr>
<tr>
<td>getMarkerName</td>
<td>Retrieves the name of the marker at the specified index.</td>
</tr>
<tr>
<td>getMarkerTime</td>
<td>Retrieves the time of the marker at the specified index.</td>
</tr>
<tr>
<td>isIdentical</td>
<td>Retrieves a value indicating whether the supplied object is the same as the current one.</td>
</tr>
<tr>
<td>isMemberOf</td>
<td>Retrieves a value indicating whether the specified media item is a member of the specified playlist.</td>
</tr>
<tr>
<td>isReadOnlyItem</td>
<td>Retrieves a value indicating whether the attributes of the specified media item can be edited.</td>
</tr>
<tr>
<td>setItemInfo</td>
<td>Sets the value of the specified attribute for the media item.</td>
</tr>
</tbody>
</table>
The **Media** object is accessed through the following properties and methods.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property or method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Controls</strong></td>
<td><code>currentItem</code></td>
</tr>
<tr>
<td><strong>Player</strong></td>
<td><code>currentMedia, newMedia</code></td>
</tr>
<tr>
<td><strong>Playlist</strong></td>
<td><code>item</code></td>
</tr>
</tbody>
</table>

Because it is the most common means of access, `player.currentMedia` is used for purposes of illustration in the reference syntax sections.

**See Also**

- [Object Model Reference](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Media.attributeCount**

The **attributeCount** property retrieves the number of attributes that can be queried and/or set for the media item.

**Syntax**

`player.currentMedia.attributeCount`

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

**Example Code**
The following JScript example uses `Media.attributeCount` to determine the number of attributes available in the current media item. The code uses that value to print a list of attribute names and values in an HTML text area, named myText. The player object was created with ID = "Player".

```javascript
// Store the current media object.
var cm = Player.currentMedia;

// Create arrays to hold each attribute name and value.
var atNames = new Array();
var atValues = new Array();

// Loop through the attribute list.
for(var i = 0; i < cm.attributeCount; i++) {
    // Fill the arrays with the attribute info.
    atNames[i] = cm.getAttributeName(i);
    atValues[i] = cm.getItemInfo(atNames[i]);

    // Print the attribute information to the text area.
    myText.value += atNames[i] + " : " + atValues[i];
    myText.value += "\n";
}
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in `wmp.idl`; include `wmp.h`.

**Library:** Use `wmp.dll`.

See Also

- Media Object
- Media.getAttributeName
- Media.getItemInfo
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.

Media.duration

The `duration` property retrieves the duration of the current media item in seconds.
Syntax

`player.currentMedia.duration`

**Possible Values**

This property is a read-only `Number (double)`.

**Remarks**

If this property is used with a media item other than the one specified in `Player.currentMedia`, it may not contain a valid value.

To retrieve the value of this property, read access to `Media Library` is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses `Media.duration` to display the time remaining in the current media item. An HTML DIV element named RemTime displays the information. An HTML timer updates the text in the DIV element every second. The following JScript code starts the timer:

```javascript
// Execute the update() function at one-second intervals.
idTmr = window.setInterval("update()",-1000);
```

The following JScript code stops the timer:

```javascript
window.clearInterval(idTmr);
```

Use the `Player.PlayStateChange` event with a `switch` statement to determine when to start and stop the timer.

The following JScript code executes each time the timer calls the update function:

```javascript
// Store the current position of the current media item.
var TimeNow = Player.controls.currentPosition;

// Display the time remaining information.
RemTime.innerHTML = "Seconds remaining: ";

// Subtract the current position from the duration of the current media.
// Use the Math.floor method to round the result down to the nearest integer.
RemTime.innerHTML += Math.floor(Player.currentMedia.duration - TimeNow);
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Media Object](#)
Media.durationString

The **durationString** property retrieves a **String** value indicating the duration of the current media item in HH:MM:SS format.

**Syntax**

```javascript
player.currentMedia.durationString
```

**Possible Values**

This property is a read-only **String**.

**Remarks**

If this property is used with a media item other than the one specified in **Player.currentMedia**, it may not contain a valid value. If the media item is less than an hour long, the HH: portion of the return value is omitted.

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses **Media.durationString** to display the duration of the current media item as formatted text. An HTML DIV element named MediaInfo displays the duration information. The player object was created with ID = "Player".

```javascript
<!-- Create an event handler to update the display when
the current media item changes. -->
<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = OpenStateChange(NewState)>

// Test whether the new media item is open.
if (NewState == 13){

    // Write the formatted duration string to the DIV region.
    MediaInfo.innerHTML = "Duration: " + Player.currentMedia.durationString;

</SCRIPT>
```
Media.error

The `error` property retrieves an `ErrorItem` object if the media item has an error condition.

**Syntax**

```
player.currentMedia.error
```

**Possible Values**

This property is a read-only `ErrorItem` object.

**Remarks**

If the media item cannot be played, this property retrieves an `ErrorItem` object that contains information about the problem encountered.

**Requirements**

**Version:** Windows Media Player for Windows XP or later.
Media.getAttributeCountByType

The `getAttributeCountByType` method retrieves the number of attributes associated with the specified attribute name and language.

Syntax

```
player.currentMedia.getAttributeCountByType(name, language)
```

Parameters

- **name**

  String containing the name of the attribute. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

- **language**

  String representing the language. If the value is set to null or "" (empty string) the current locale string is used. Otherwise, the value must be a valid RFC 1766 language string such as "en-us".

Return Values

This method returns a **Number (long)** containing the attribute count.

Remarks

This method is used to determine the number of attributes corresponding to a particular attribute name for a given Media object. Index numbers can then be passed to the `getItemInfoByType` method. This is useful, for example, when a media item has been categorized under multiple genres.
To use this method, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Media Object
- Media.getItemInfoByType
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

Media.getAttributeName

The getAttributeName method retrieves the name of the attribute corresponding to the specified index.

Syntax

player.currentMedia.getAttributeName(index)

Parameters

index

Number (long) containing the index of the attribute.

Return Values

This method returns a String specifying the name of the attribute.

Remarks

The attribute name returned can be used in conjunction with getItemInfo to retrieve the value for a specific
named attribute.

To use this method, read access to Media Library is required. For more information, see Media Library Access.

For information about the attributes supported by Windows Media Player, see Available Attributes.

Example Code

The following JScript example uses Media.getAttributeName to fill an HTML text area named myText with the index and name of each attribute for the current media item. The player object was created with ID = "Player".

```javascript
// Store the current media object.
var cm = Player.currentMedia;

// Get the number of attributes for the current media.
var atCount = cm.attributeCount;

// Loop through the attribute list.
for(var i=0; i < atCount; i++){
    // Print each attribute index and name.
    myText.value += "Attribute " + i + ": " + cm.getAttributeName(i) + "\n";
}
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Media Object
- Media.getAttributeCount
- Media.getAttributeCount
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.
Media.getItemInfo

The **getItemInfo** method retrieves the value of the specified attribute for the current media item.

**Syntax**

```
player.currentMedia.getItemInfo(name)
```

**Parameters**

*name*

**String** containing the name of the attribute. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

**Return Values**

This method returns a **String** representing the value of the specified attribute. For attributes whose underlying value is **Boolean**, it returns the string "true" or "false".

**Remarks**

This method is used to retrieve metadata information about a specific piece of media content, whether it stands alone or is part of a playlist.

The **attributeCount** property can be used to determine the number of attribute names available for a given **Media** object. Index numbers can then be used with the **getAttributeName** method to determine the names of the attributes, which can in turn be passed to **getItemInfo**.

To retrieve attributes with multiple values and attributes with complex values, use the **getItemInfoByType** method.

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Media Object**
- **Media.attributeCount**
- **Media.getAttributeName**
- **Media.getItemInfoByType**
- **Media.setItemInfo**
Media.getItemInfoByAtom

The `getItemInfoByAtom` method retrieves the value of the attribute with the specified index number.

Syntax

```
player.currentMedia.getItemInfoByAtom(index)
```

Parameters

- `index`  
  Number (`long`) specifying the index at which a given attribute resides within the set of available attributes.

Return Values

This method returns a `String` representing the value of the specified attribute. For attributes whose underlying value is `Boolean`, it returns the string "true" or "false".

Remarks

This method can be used to retrieve metadata information about a specific piece of media using an index number. The `attributeCount` property can be used to determine the number of attributes available for the media item. The `getMediaAtom` method of the `MediaCollection` object can also be used to retrieve the index of a particular media library attribute. This technique is generally more efficient than the `getItemInfo` or `getItemInfoByType` methods when working with large playlists.

To use this method, read access to `Media Library` is required. For more information, see `Media Library Access`.

For information about the attributes supported by Windows Media Player, see `Available Attributes`.

See Also

- `Media Object`
- `Media.attributeCount`
Media.getItemInfo
Media.getItemInfoByType
Media.setItemInfo
MediaCollection.getMediaAtom
Reading Attribute Values
Settings.mediaAccessRights
Settings.requestMediaAccessRights

Media.getItemInfoByType

The `getItemInfoByType` method retrieves the value of the attribute corresponding to the specified attribute name, language, and index.

**Syntax**

```plaintext
player.currentMedia.getItemInfoByType(name, language, index)
```

**Parameters**

- **name**

  String containing the name of the attribute. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

- **language**

  String representing the language. If the value is set to null or "" (empty string) the current locale string is used. Otherwise, the value must be a valid RFC 1766 language string such as "en-us".

- **index**

  Number (long) containing the zero-based index of the attribute.

**Return Values**

This method returns a String, MetadataPicture object, or MetadataText object as indicated in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Return value</th>
</tr>
</thead>
</table>
For attributes whose underlying value is **Boolean**, it returns the string "true" or "false".

**Remarks**

This method is used to retrieve metadata information about a specific piece of media content, whether it stands alone or is part of a playlist.

This method supports attributes with multiple values and attributes with complex values, which the `getItemInfo` method does not support.

The `attributeCount` property can be used to determine the number of attributes available for a given `Media` object. Index numbers can then be used with the `getAttributeName` method to determine the names of the attributes, which can in turn be passed to the `name` parameter of `getItemInfoByType`.

The `getAttributeCountByType` method can be used to determine the number of attributes corresponding to a particular attribute name for a given `Media` object. Index numbers can then be passed to the `index` parameter of `getItemInfoByType`. This is useful, for example, when a media item has been categorized under multiple genres.

To use this method, read access to `Media Library` is required. For more information, see [Media Library Access](#).

This method can throw errors. You should include error handling code when you call this method. For example, in JScript you can implement error handling using the `try...catch...finally` structure.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Media Object](#)
- [Media.attributeCount](#)
- [Media.getAttributeCountByType](#)
- [Media.getAttributeName](#)
- [Media getItemInfo](#)
- [Media.setItemInfo](#)
- [MetadataPicture Object](#)
- [MetadataText Object](#)
- [Reading Attribute Values](#)
- [Settings.mediaAccessRights](#)
Media.getMarkerName

The `getMarkerName` method retrieves the name of the marker at the specified index.

Syntax

```javascript
player.currentMedia.getMarkerName(markerNum)
```

Parameters

- `markerNum`

`Number (long)` specifying a marker index.

Return Values

This method returns a `String`.

Remarks

This method returns `NULL` if the specified marker does not exist.

Some media clips do not contain markers. Use `markerCount` to find out how many markers are in the current clip.

Marker index numbers start at 1.

To use this method, read access to `Media Library` is required. For more information, see `Media Library Access`.

Example Code

The following JScript example uses `Media.getMarkerName` to fill an HTML TEXTAREA element named `MNAMES` with the names of the markers in the current media item. The player object was created with `ID = "Player"`.

```javascript
// Get the number of markers in the current media.
var mcount = Player.currentMedia.markerCount;
```
// Verify that at least one marker exists in the current media.
if (mcount > 0) {

// Loop through the marker list.
for (var i = 1; i < mcount + 1; i++) {

    // Print the marker name to the text area.
    MNAMES.value += Player.currentMedia.getMarkerName(i);
    MNAMES.value += "\n";
}

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Media Object
- Media.getMarkerTime
- Media.markerCount
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.

Media.getMarkerTime

The getMarkerTime method retrieves the time of the marker at the specified index.

Syntax

player.currentMedia.getMarkerTime(markerNum)

Parameters

markerNum

Number (long) specifying the marker index.
**Return Values**

This method returns a **Number (double)** specifying the location of the marker in seconds from the beginning of the clip.

**Remarks**

This method returns **NULL** if the specified marker does not exist.

Some media clips do not contain markers. Use **markerCount** to find out how many markers are in the current clip.

Marker index numbers start at 1.

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses **Media.getMarkerTime** to fill an HTML TEXTAREA element named MTIMES with the location of each marker. The player object was created with ID = "Player".

```js
// Get the number of markers in the current media.
var mcount = Player.currentMedia.markerCount;

// Verify that at least one marker exists in the current media.
if (mcount > 0){

// Loop through the marker list.
for (var i = 1; i < mcount + 1; i++){

    // Print the message to the text area.
    MTIMES.value += "Marker number " + i + " occurs at ";
    MTIMES.value += Player.currentMedia.getMarkerTime(i);
    MTIMES.value += " second(s).";
    MTIMES.value += "\n";
}
}
```

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- [Media Object](#)
- [Media.getMarkerName](#)
- [Media.markerCount](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
The **ImageSourceHeight** property retrieves the height of the current media item in pixels.

**Syntax**

```
player.currentMedia.imageSourceHeight
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

If the media item is not the current one, this property returns zero.

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses **Media.imageSourceHeight** to display the image size, in pixels, of the current media item. The information is printed to an HTML TEXTAREA element named VideoSize. The player object was created with ID = "player".

```javascript
<!-- Create an event handler to refresh the information when the current media changes. --
<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = OpenStateChange(NewState)>

// Test whether the new media item is open.
if (NewState == 13) {

    // Store the height and width of the new media item.
    var Height = Player.currentMedia.imageSourceHeight;
    var Width = Player.currentMedia.imageSourceWidth;

    // Erase the information in the text area.
    VideoSize.value = "";

    // Test whether an image is visible.
    if (Height != 0 && Width != 0)

        // Display the image size information.
        VideoSize.value = Width + " x " + Height;

</SCRIPT>
```
Media.imageSourceWidth

The `imageSourceWidth` property retrieves the width of the current media item in pixels.

Syntax

```
player.currentMedia.imageSourceWidth
```

Possible Values

This property is a read-only `Number (long)`.

Remarks

If the media item is not the current one, this property returns zero.

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses `Media.imageSourceWidth` to display the image size, in pixels, of the
current media item. The information is printed to an HTML TEXTAREA element named VideoSize. The player object was created with ID = "player".

<!-- Create an event handler to refresh the information when the current media changes. --
<SCRIPT LANGUAGE = "JScript" FOR = "Player" EVENT = OpenStateChange(NewState)>

// Test whether the new media item is open.
if (NewState == 13){

    // Store the height and width of the new media item.
    var Height = Player.currentMedia.imageSourceHeight;
    var Width = Player.currentMedia.imageSourceWidth;

    // Erase the information in the text area.
    VideoSize.value = "";

    // Test whether an image is visible.
    if (Height != 0 && Width != 0)

        // Display the image size information.
        VideoSize.value = Width + " x " + Height;
}
</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Media Object
- Player.currentMedia
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

Media.isIdentical

The isIdentical method retrieves a value indicating whether the supplied object is the same as the current one.
Syntax

\texttt{player.currentMedia.isIdentical(media)}

Parameters

\textit{media}

\textbf{Media} object to compare with the current one.

Return Values

This method returns a \textbf{Boolean}.

Example Code

The following JScript example uses \texttt{Media.isIdentical} to check whether a media item named \texttt{newMedia} is the same as the current media item. If they are not the same, the new media item is played. Otherwise, the current media continues to play uninterrupted. The player object was created with ID = "Player".

```javascript
// Check the new media item to see if it matches the current one.
if (newMedia.isIdentical(Player.currentMedia) != true){

    // Change the current media to the new media item.
    Player.currentMedia = newMedia;

    // Play the new media item.
    Player.controls.play();
}
```

Requirements

\textbf{Version}: Windows Media Player version 7.0 or later.

\textbf{Header}: Defined in wmp.idl; include wmp.h.

\textbf{Library}: Use wmp.dll.

See Also

- \textbf{Media Object}

© 2000-2003 Microsoft Corporation. All rights reserved.
Media.isMemberOf

The isMemberOf method returns a value indicating whether the media item is a member of the specified playlist.

Syntax

```javascript
player.currentMedia.isMemberOf(playlist)
```

Parameters

`playlist`

Playlist object that might contain the media item.

Return Values

This method returns a `Boolean`.

Remarks

This method is not supported with playlists retrieved through the MediaCollection object. To test whether a media item is a member of a particular named playlist, retrieve the playlist with `player.playlistCollection.getByName(name).item(0)`. This method can also be used with CD and metafile playlists.

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses `Media.isMemberOf` to test whether the current media item is a member of the playlist named Sample Playlist. If it is not, the current media item is appended to the sample playlist. The player object was created with ID = "Player".

```javascript
// Store the playlist object named Sample Playlist.
var sPlaylist = Player.playlistcollection.getbyname("Sample Playlist").item(0);

// Test whether the current media item is a member of Sample Playlist.
var answer = ((Player.currentMedia.isMemberOf(sPlaylist)) ? "Yes" : "No");

// If the current media item is not a member of Sample Playlist,
// append the current media item to the playlist.
if (answer == "No"){
    sPlaylist.appendItem(Player.currentMedia);
}
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.
Media.isReadOnlyItem

The **isReadOnlyItem** method returns a value indicating whether the specified attribute of the media item can be edited.

### Syntax

```javascript
player.currentMedia.isReadOnlyItem(attribute)
```

### Parameters

- **attribute**

  *String* indicating the name of the attribute to test. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

### Return Values

This method returns a **Boolean**.

### Remarks

If an attribute is read-only then it cannot be set with the **setItemInfo** method.

To use this method, read access to [Media Library](#) is required. For more information, see [Media Library Access](#).

### Example Code
The following JScript example uses `Media.isReadOnlyItem` to fill an HTML TEXTAREA element named `rwText` with information about the current media item. The code outputs each attribute of the current media item, along with text indicating whether the attribute is read-only or read/write. The player object was created with ID = "Player".

```javascript
// Store the current media item object.
var cm = Player.currentMedia;

// Create a variable to hold each attribute name.
var atName;

// Loop through the attribute list.
for(var i = 0; i < cm.attributeCount; i++) {
    // Get the attribute name.
    atName = cm.getAttributeName(i);

    // Test whether the attribute is read-only.
    var test = ((cm.isReadOnlyItem(atName)) ? "Read-Only" : "Read/Write");

    // Print the attribute information to the text area.
    rwText.value += atName + " is " + test + "\n";
}
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in `wmp.idl`; include `wmp.h`.

**Library:** Use `wmp.dll`.

**See Also**

- [Media Object](#)
- [Media.setItemInfo](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Media.markerCount**

The `markerCount` property retrieves the number of markers in the media item.
Syntax

player.currentMedia.markerCount

Possible Values

This property is a read-only Number (long) specifying the number of markers in the file.

Remarks

This property returns zero if a file has no markers, or if the media item is not the same as Player.currentMedia. Marker numbers start at 1.

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses Media.markerCount to retrieve the number of markers in the current media item. That value is then used as the upper boundary for a looping structure, which iterates through the marker list to retrieve each marker name. An HTML TEXTAREA element named MNAMES displays the names of the markers in the current media item. The player object was created with ID = "Player".

```javascript
// Get the number of markers in the current media item.
var mcount = Player.currentMedia.markerCount;

// Verify that at least one marker exists in the current media item.
if (mcount > 0) {

    // Loop through the marker list.
    for (var i = 1; i < mcount + 1; i++) {

        // Print the marker name to the text area.
        MNAMES.value += Player.currentMedia.getMarkerName(i);
        MNAMES.value += "\n";
    }
}
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Media Object
- Player.currentMedia
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

Previous Next
Media.name

The **name** property specifies or retrieves the name of the media item.

**Syntax**

```javascript
player.currentMedia.name
```

**Possible Values**

This property is a read/write **String** containing the name of the media item.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. To specify the value of this property, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Remarks**

Before using this method to specify the name of a media item, use **isReadOnlyItem** to determine whether the name can be set.

**Example Code**

The following JScript example uses **Media.name** to change the name of the current media item. An HTML TEXT input element named NameText allows the user to enter a text string for the new name. The player object was created with ID = "Player".

```html
<INPUT type = "BUTTON"  id = "NewName"  name = "NewName"  value = "Change Name" onClick = "
        /* Store the current media item. */
        var cm = Player.currentMedia;

        /* Change the name. */
        cm.name = NameText.value;
"
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.
Media.setItemInfo

The `setItemInfo` method sets the value of the specified attribute for the current media item.

**Syntax**

```javascript
player.currentMedia.setItemInfo(attribute, value)
```

**Parameters**

- `attribute`

  *String* containing the attribute name. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

- `value`

  *String* containing the new value.

**Return Values**

This method does not return a value.

**Remarks**

The `attributeCount` property can be used to determine the number of attributes available for a given `Media` object. Index numbers can then be used with the `getAttributeName` method to determine the names of the built-in attributes that can be used with this method.

Before using this method, use the `isReadOnlyItem` method to determine whether a particular attribute can be set.
To use this method, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Note** If you embed the control in your application, file attributes that you change will not be written to the media file itself until the user runs Windows Media Player. If you use the control in a remoted application written in C++, file attributes that you change will be written to the media file itself shortly after you make the changes. In either case, the changes are immediately available to you through Media Library.

**Example Code**

The following JScript example uses `Media.setItemInfo` to change the value of the Genre attribute for the current media item. An HTML TEXT input element named genText allows the user to enter a text string, which is then used to change the attribute information. The player object was created with ID = "Player".

```plaintext
<!-- Create the button element. -->
<input type = "BUTTON"  id = "NEWGEN"  name = "NEWGEN"  value = "Change Genre"
onClick = " /* Store the current media item. */
var cm = Player.currentMedia;

/* Get the user input from the text box. */
var atValue = genText.value;

/* Test for read-only status of the attribute. */
if(cm.isReadOnlyItem('Genre') == false){
  /* Change the attribute value. */
  cm.setItemInfo('Genre' ,atValue);
}
"

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Media Object](#)
- [Media.getItemInfo](#)
- [Media.isReadOnlyItem](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Media.sourceURL

The **sourceURL** property retrieves the URL of the media item.

**Syntax**

```
player.currentMedia.sourceURL
```

**Possible Values**

This property is a read-only **String**.

**Example Code**

The following JScript example uses **Media.sourceURL** to retrieve the URL of the first media item in the sample playlist; the URL of the media item is then assigned to the player object **URL** property. The player object was created with ID = "Player".

```javascript
// Store the sample playlist object in a variable.
var pl = Player.playlistCollection.getByName("Sample Playlist").item(0);

// Store the first media item from the sample playlist.
var media = pl.item(0);

// Change the URL property of the player to the URL of the media item.
Player.URL = media.sourceURL;

// Play the media item.
Player.controls.play();
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Media Object](#)
MediaCollection Object

The MediaCollection object provides a way to organize a large collection of media items. It can be queried to generate playlists automatically.

The MediaCollection object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Adds a new media item or playlist to Media Library.</td>
</tr>
<tr>
<td>getAll</td>
<td>Retrieves a Playlist object containing all media items in Media Library.</td>
</tr>
<tr>
<td>getAttributeStringCollection</td>
<td>Retrieves a StringCollection object representing the set of all values for a given attribute within a given media type.</td>
</tr>
<tr>
<td>getByAlbum</td>
<td>Retrieves a Playlist object containing media items from the specified album.</td>
</tr>
<tr>
<td>getByAttribute</td>
<td>Retrieves a Playlist object containing media items with the specified attribute having the specified value.</td>
</tr>
<tr>
<td>getByAuthor</td>
<td>Retrieves a Playlist object containing media items by the specified author.</td>
</tr>
<tr>
<td>getByGenre</td>
<td>Retrieves a Playlist object containing media items with the specified genre.</td>
</tr>
<tr>
<td>getName</td>
<td>Retrieves a Playlist object containing media items with the specified name.</td>
</tr>
<tr>
<td>getMediaAtom</td>
<td>Retrieves the index at which a given property resides within the set of available properties.</td>
</tr>
<tr>
<td>isDeleted</td>
<td>Retrieves a value indicating whether the specified media item is in the deleted items folder.</td>
</tr>
<tr>
<td>remove</td>
<td>Removes an item from the media collection.</td>
</tr>
<tr>
<td>setDeleted</td>
<td>Moves the specified media item to the deleted items folder.</td>
</tr>
</tbody>
</table>

The MediaCollection object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>mediaCollection</td>
</tr>
</tbody>
</table>
MediaCollection.add

The add method adds a new media item or playlist to Media Library.

Syntax

player.mediaCollection.add(path)

Parameters

path

String containing the path.

Return Values

This method returns a Media object.

Remarks

This method loads an existing media item or playlist into Media Library, given a path. This method does not move or change the file. This method fails if given an invalid local path, but media items themselves are not checked for validity before they are added to Media Library.

This method accepts both static and auto playlist files. The PlaylistCollection.importPlaylist method can also be used to add a static playlist to Media Library.

To use this method, full access to Media Library is required. For more information, see Media Library Access.

Example Code

The following Microsoft JScript example adds three media objects to the player media collection. The player object was created with ID="Player".

// Adding a media object using a Web site.
Player.mediaCollection.add("http://www.proseware.com/Media/Laure.wma");
Adding a media object from a local network.
You must add an escape sequence of a backslash for every original backslash.
Player.mediaCollection.add("\\\yourservername\Public\Jeanne.wma");

Adding a media object from a file on a local drive.
Be sure to add appropriate escape sequences.
Player.mediaCollection.add("C:\WMSDK\WMPSDK\docs\samples\media\house.wma");

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Managing Playlists
- Media Object
- MediaCollection Object
- MediaCollection.remove
- PlaylistCollection.importPlaylist
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

Previous Next

© 2000-2003 Microsoft Corporation. All rights reserved.

MediaCollection.getAll

The getAll method retrieves a playlist containing all media items in Media Library.

Syntax

player.mediaCollection.getAll()

Parameters

This method takes no parameters.

Return Values
This method returns a **Playlist** object.

**Remarks**

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses *MediaCollection.getAll* to play media items randomly from the media collection. The Player object was created with ID = "Player".

```javascript
// Store the count of all media items in the media collection.
var count = Player.mediaCollection.getAll().count;

// Generate a random number using the media count.
var rand = Math.random() * count;

// Round down the random number to the nearest integer.
rand = Math.floor(rand);

// Make the random media item the current media item.
Player.currentMedia = Player.mediaCollection.getAll().item(rand);

// Play the media item.
// Player.controls.play();
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [MediaCollection Object](#)
- [Playlist Object](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
MediaCollection.getAttributeStringCollection

The `getAttributeStringCollection` method retrieves a `StringCollection` object representing the set of all values for a given attribute within a given media type.

Syntax

`player.mediaCollection.getAttributeStringCollection(attribute, mediaType)`

Parameters

- **attribute**
  
  String specifying the attribute.

- **mediaType**
  
  String representing the media type. Contains one of the following values: "Audio", "Video", "Playlist", or "Other".

Return Values

This method returns a `StringCollection` object.

Remarks

The following table lists the attributes supported by this method for each media type when using Windows Media Player 9 Series.

<table>
<thead>
<tr>
<th>Audio</th>
<th>Video</th>
<th>Playlist</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Actor</td>
<td>Genre</td>
<td>Genre</td>
</tr>
<tr>
<td>Album</td>
<td>Artist</td>
<td>WM/Genre</td>
<td>WM/Genre</td>
</tr>
<tr>
<td>AlbumArtist</td>
<td>Author</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlbumArtistSortOrder</td>
<td>AuthorSortOrder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlbumID</td>
<td>Director</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlbumIDAlbumArtist</td>
<td>Genre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlbumTitleSortOrder</td>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artist</td>
<td></td>
<td>Language</td>
<td></td>
</tr>
<tr>
<td>Author</td>
<td></td>
<td>ProducedBy</td>
<td></td>
</tr>
<tr>
<td>AuthorSortOrder</td>
<td>ReleasedBy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

For information about the attributes supported by Windows Media Player, see [Media Item Attributes](#).

**Example Code**

The following JScript example uses `MediaCollection.getAttributeStringCollection` to display a list of values that correspond to a particular attribute for audio media in the media collection. An HTML SELECT element, created with ID = "Attribute", allows the user to select an attribute, such as Artist, Genre, or Album. An HTML TEXTAREA element, created with ID = "AttributeVals", displays the result. The Player object was created with ID = "Player".

```javascript
// Clear the text in the display area.
AttributeVals.value = "";

// Store the mediaCollection object.
var library = Player.mediaCollection;

// Get the string collection for the attribute type the user selects.
var all = library.getAttributeStringCollection(Attribute.value, "Audio");

// Loop through the string collection.
for (i = 0; i < all.count; i++){
    // Display the items one line at a time.
    AttributeVals.value += all.item(i);
    AttributeVals.value += "\n";
}
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**
MediaCollection.getObject

Settings.mediaAccessRights
Settings.requestMediaAccessRights
StringCollection Object

MediaCollection.getByAlbum

The `GetByAlbum` method retrieves a playlist containing the media items from the specified album.

**Syntax**

```
player.mediaCollection.getByAlbum(album)
```

**Parameters**

* album

String containing the name of the album.

**Return Values**

This method returns a Playlist object.

**Remarks**

To use this method, read access to Media Library is required. For more information, see Media Library Access.

**Example Code**

The following JScript example uses `MediaCollection.getByAlbum` to retrieve a playlist of media items. The playlist contains items with the album specified by the user in an HTML TEXT input element named GetAlbum. The Player object was created with ID = "Player".

```html
<!-- Create an HTML BUTTON element to create the playlist and play the media. -->
<input type = "button" name = "PlayAlbum" id = "PlayAlbum" value = "Play Album"
onClick = "/* Retrieve the album title text from the user. */
var album = GetAlbum.value;

/* Create the playlist using getByAlbum. */
```
var pl = Player.mediaCollection.getByAlbum(album);

/* Make the new playlist the current playlist. */
Player.currentPlaylist = pl;

/* Play the media in the new playlist. */
Player.controls.play();

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection Object
- Playlist Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

MediaCollection.getByAttribute

The `getByAttribute` method retrieves a playlist of media items with the specified attribute having the specified value.

Syntax

`player.mediaCollection.getByAttribute(attribute, value)`

Parameters

attribute

String indicating the name of the attribute to search on. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

value
**String** indicating the value that the attribute should have.

**Return Values**

This method returns a **Playlist** object.

**Remarks**

This method can be used to create a generic query for media items that match a value for an attribute in the database. This is especially useful in the case of user-defined attributes. If the attribute does not exist, an error will result.

You can use this method to retrieve all of the media items of a specific type. Use the attribute name "MediaType" and one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>Music and other audio-only items.</td>
</tr>
<tr>
<td>playlist</td>
<td>Playlist items. You can also retrieve these items using <strong>PlaylistCollection.getAll</strong>.</td>
</tr>
<tr>
<td>radio</td>
<td>Radio station items.</td>
</tr>
<tr>
<td>video</td>
<td>Video items.</td>
</tr>
</tbody>
</table>

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses **MediaCollection.getByAttribute** to play all content from **Media Library** by the artist named Triode 48. The Player object was created with ID = "Player".

```javascript
// Get a playlist object filled with media items by a particular artist.
var pl = Player.mediaCollection.getByAttribute("Artist", "Triode 48");

// Make the new playlist the current one.
Player.currentPlaylist = pl;

// Start the player.
Player.controls.play();
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [MediaCollection Object](#)
MediaCollection.getByAuthor

The `getByAuthor` method retrieves a playlist of the media items by the specified author.

Syntax

```javascript
player.mediaCollection.getByAuthor(author)
```

Parameters

- `author` String containing the name of the author.

Return Values

This method returns a `Playlist` object.

Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses `MediaCollection.getByAuthor` to retrieve a playlist of media items. The playlist contains items matching the author specified by the user in an HTML TEXT input element named GetAuthor. The Player object was created with ID = "Player".

```javascript
<!-- Create an HTML BUTTON element to create the playlist and play the media item. -->
<INPUT TYPE = "BUTTON"  NAME = "PlayAuthor"  ID = "PlayAuthor"  VALUE = "Play Author"
onClick = "
    /* Retrieve the author name text from the user. */
    var author = GetAuthor.value;
    /* Create the playlist using getByAuthor. */
    var pl = Player.mediaCollection.getByAuthor(Author);
";
```
/* Make the new playlist the current playlist. */
Player.currentPlaylist = pl;

/* Play the media item in the new playlist. */
Player.controls.play();

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection Object
- Playlist Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

MediaCollection.getByGenre

The **getByGenre** method retrieves a playlist of the media items with the specified genre.

Syntax

```plaintext
player.mediaCollection.getByGenre(genre)
```

Parameters

- **genre**
  - String containing the name of the genre.

Return Values

This method returns a **Playlist** object.
Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses MediaCollection.getByGenre to retrieve a playlist of media items. The playlist contains items with the genre specified by the user in an HTML TEXT input element named GetGenre. The Player object was created with ID = "Player".

```html
<!-- Create an HTML BUTTON element to create the playlist and play the media. -->
<input type = "BUTTON"  name = "PlayGenre"  id = "PlayGenre"  value = "Play Genre"
onclick = "
    /* Retrieve the genre text from the user. */
    var genre = GetGenre.value;

    /* Create the playlist using getByGenre. */
    var pl = Player.mediaCollection.getByGenre(genre);

    /* Make the new playlist the current playlist. */
    Player.currentPlaylist = pl;

    /* Play the media item in the new playlist. */
    Player.controls.play();
">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection Object
- Playlist Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.
MediaCollection.getByName

The `getByName` method retrieves a playlist of the media items with the specified name.

Syntax

```
player.mediaCollection.getByName(name)
```

Parameters

- `name`  
  String containing the name.

Return Values

This method returns a Playlist object.

Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses `MediaCollection.getByName` to retrieve three items from the Media Library. Each item is then appended to the current playlist. The Player object was created with ID="Player".

```
// In each case, use the name exactly as it appears in Media Library.
// Windows Media Player does not include file extensions or file paths
// in the name. Internet URL's include the entire path, but not the
// file extension.

// Get a playlist object which contains an Internet URL.
var One = Player.mediaCollection.getByName("http://www.proseware.com/Media/Laure");

// Get a playlist object which contains a file on a network server.
var Two = Player.mediaCollection.getByName("Jeanne");

// Get a playlist object which contains a file on a local drive.
var Three = Player.mediaCollection.getByName("house");

// Append each item to the current playlist. Since each playlist retrieved
// using `getByName` contains one media item, use `Playlist.item` with an
// index of zero to reference that item.
Player.currentPlaylist.appendItem(One.item(0));
Player.currentPlaylist.appendItem(Two.item(0));
Player.currentPlaylist.appendItem(Three.item(0));
```

Requirements

- **Version:** Windows Media Player version 7.0 or later.
- **Header:** Defined in wmp.idl; include wmp.h.
MediaCollection.getMediaAtom

The `getMediaAtom` method retrieves the index at which a given attribute resides within the set of available attributes.

**Syntax**

```javascript
player.mediaCollection.getMediaAtom(attribute)
```

**Parameters**

- `attribute` 

  *String* containing the attribute name. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

**Return Values**

This method returns a `Number` (`long`).

**Remarks**

The index number retrieved with this method can be passed to the `Media.getItemInfoByAtom` method to access attribute values. This technique is generally more efficient when working with large playlists than accessing an attribute value by name using `Media.getItemInfo` or `Media.getItemInfoByType`.

To use this method, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**
**MediaCollection.isDeleted**

The **isDeleted** method retrieves a value indicating whether the specified media item is in the deleted items folder.

**Syntax**

```plaintext
player.mediaCollection.isDeleted(item)
```

**Parameters**

- **item**

**Media** object that might be deleted.

**Return Values**

This method returns a **Boolean**.

**Remarks**

To use this method, read access to **Media Library** is required. For more information, see **Media Library Access**.
Example Code

The following JScript example uses `MediaCollection.isDeleted` to test whether a particular media item, stored in the variable named `mediaObject`, is in the deleted items folder. If the media item is not deleted already, then it is moved to the deleted items folder. The Player object was created with ID = "Player".

```javascript
// Test whether the media item is in the deleted items folder.
if (!Player.mediaCollection.isDeleted(mediaObject)){
    // The media item is available to be deleted, move it to
    // the deleted items folder.
    Player.mediaCollection.setDeleted(mediaObject, true);

    // Inform the user that the operation succeeded.
    alert("Media item moved to deleted items folder.");
}
else  // Tell the user the operation is unnecessary.
    alert("Media item is already deleted!");
}
```

Requirements

**Version:** Windows Media Player version 7.0, Windows Media Player version 7.1, or Windows Media Player for Windows XP. This method is not supported for Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- **Media Object**
- **MediaCollection Object**
- **MediaCollection.setDeleted**
- **Settings.mediaAccessRights**
- **Settings.requestMediaAccessRights**

© 2000-2003 Microsoft Corporation. All rights reserved.
Syntax

`player.mediaCollection.remove(item, delete)`

Parameters

`item`

`Media` object to be removed.

`delete`

`Boolean` indicating whether to remove the media item.

Return Values

This method does not return a value.

Remarks

This method deletes an item from `Media Library`. This method does not delete files from the user's hard disk.

To use this method, full access to `Media Library` is required. For more information, see [Media Library Access](#).

Example Code

The following JScript example, after prompting the user, permanently deletes the first media item in the media collection using `MediaCollection.remove`. The Player object was created with ID = "Player".

```javascript
// Retrieve the first item from the media collection.
var mediaObject = Player.mediaCollection.getAll().item(0);

// Store the name of the retrieved object.
var mediaName = mediaObject.name;

// Prompt the user for permission to delete the object.
var answer = confirm("OK to permanently delete " + mediaName + "?");

// Check the user response.
if (answer){

    // Permanently delete the item.
    Player.mediaCollection.remove(mediaObject, true);

    // Report that the item was deleted.
    alert("Deleted item " + mediaName);
}
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.
MediaCollection.setDeleted

The setDeleted method moves the specified media item to the deleted items folder.

Syntax

player.mediaCollection.setDeleted(item, true)

Parameters

item

Media object being moved.

true

Always specify this value.

Return Values

This method does not return a value.

Remarks

This method does not remove files from the user's hard disk.

To use this method, full access to Media Library is required. For more information, see Media Library Access.

Example Code

The following JScript example uses MediaCollection.setDeleted to move a particular media item, stored in the
variable named mediaObject, to the deleted items folder. The `MediaCollection.isDeleted` method first tests whether the media item is already deleted. The Player object was created with ID = "Player".

```javascript
// Test whether the media item is in the deleted items folder.
if (!Player.mediaCollection.isDeleted(mediaObject)){
    // The media item is available to be deleted; move it to
    // the deleted items folder.
    Player.mediaCollection.setDeleted(mediaObject, true);

    // Inform the user that the operation succeeded.
    alert("Media item moved to deleted items folder.");}
else
    // Tell the user the operation is unnecessary.
    alert("Media item is already deleted!");
}
```

Requirements

**Version:** Windows Media Player version 7.0, Windows Media Player version 7.1, or Windows Media Player for Windows XP. This method is not supported for Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Media Object](#)
- [MediaCollection Object](#)
- [MediaCollection.isDeleted](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

MetadataPicture Object

The `MetadataPicture` object provides a way to retrieve the values of the `WM/Picture` metadata attribute. This attribute corresponds to album art contained in a digital media file, not to album art downloaded over the Internet.

The `MetadataPicture` object supports the following properties.
The `MetadataPicture` object is accessed through the following method.

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Media</code></td>
<td><code>getItemInfoByType</code></td>
</tr>
</tbody>
</table>

For purposes of illustration, `player.currentMedia.getItemInfoByType(name, language, index)` is used in the reference syntax sections.

See Also

- [Object Model Reference](#)

**MetadataPicture.description**

The `description` property retrieves a description of the metadata image.

**Syntax**

```javascript
player.currentMedia.getItemInfoByType(name, language, index).description
```

**Possible Values**

This property is a read-only `String`.

**Remarks**

To retrieve the value of this property, read access to `Media Library` is required. For more information, see [Media Library Access](#).
MetadataPicture.mimeType

The **mimeType** property retrieves the standard MIME type of the metadata image.

**Syntax**

```javascript
player.currentMedia.getItemInfoByType(name, language, index).mimeType
```

**Possible Values**

This property is a read-only **String**.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. For more information, see **Media Library Access**.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.
MetadataPicture.pictureType

The `pictureType` property retrieves the picture type of the metadata image.

Syntax

```c
player.currentMedia.getItemInfoByType(name, language, index).pictureType
```

Possible Values

This property is a read-only `String`.

Remarks

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MetadataPicture Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
MetadataPicture.URL

The `URL` property retrieves the URL of the metadata image.

**Syntax**

```javascript
player.currentMedia.getItemInfoByType(name, language, index).URL
```

**Possible Values**

This property is a read-only `String`.

**Remarks**

To retrieve the value of this property, read access to `Media Library` is required. For more information, see `Media Library Access`.

**Requirements**

- **Version**: Windows Media Player 9 Series or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- [MetadataPicture Object](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
MetadataText Object

The MetadataText object provides a way to retrieve metadata for complex textual metadata attributes.

The MetadataText object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Retrieves a description of the metadata text.</td>
</tr>
<tr>
<td>text</td>
<td>Retrieves the metadata text.</td>
</tr>
</tbody>
</table>

The MetadataText object is accessed through the following method.

Object          Method
Media           getItemInfoByType

For purposes of illustration, player.currentMedia.getItemInfoByType(name, language, index) is used in the reference syntax sections.

See Also

- Object Model Reference

MetadataText.description

The description property retrieves a description of the metadata text.

Syntax

player.currentMedia.getItemInfoByType(name, language, index).description

Possible Values

This property is a read-only String.
Remarks

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MetadataText Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.

MetadataText.text

The text property retrieves the metadata text.

Syntax

player.currentMedia.getItemInfoByType(name, language, index).text

Possible Values

This property is a read-only String.

Remarks

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player 9 Series or later.
Network Object

The Network object provides properties and methods used to access statistics relating to the quality of a network connection, and to specify and retrieve the network proxy settings.

The Network object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth</td>
<td>Retrieves the current bandwidth of the clip.</td>
</tr>
<tr>
<td>bitRate</td>
<td>Retrieves the current bit rate being received.</td>
</tr>
<tr>
<td>bufferingCount</td>
<td>Retrieves the number of times buffering occurred during clip playback.</td>
</tr>
<tr>
<td>bufferingProgress</td>
<td>Retrieves the percentage of buffering completed.</td>
</tr>
<tr>
<td>bufferingTime</td>
<td>Specifies or retrieves the amount of buffering time in milliseconds before playing begins.</td>
</tr>
<tr>
<td>downloadProgress</td>
<td>Retrieves the percentage of download completed.</td>
</tr>
<tr>
<td>encodedFrameRate</td>
<td>Retrieves the video frame rate specified by the content author.</td>
</tr>
<tr>
<td>frameRate</td>
<td>Retrieves the current video frame rate.</td>
</tr>
<tr>
<td>framesSkipped</td>
<td>Retrieves the total number of frames skipped during playback.</td>
</tr>
<tr>
<td>lostPackets</td>
<td>Retrieves the number of packets lost.</td>
</tr>
<tr>
<td>maxBandwidth</td>
<td>Specifies or retrieves the maximum allowed bandwidth.</td>
</tr>
<tr>
<td>maxBitRate</td>
<td>Retrieves the maximum possible video bit rate.</td>
</tr>
</tbody>
</table>
The **Network** object supports the following methods.

**Method** | **Description**
--- | ---
**getProxyBypassForLocal** | Retrieves a value indicating whether the proxy server should be bypassed if the origin server is on a local network.
**getProxyExceptionList** | Retrieves the proxy exception list.
**getProxyName** | Retrieves the name of a proxy server to use.
**getProxyPort** | Retrieves the proxy port to use.
**getProxySettings** | Retrieves the proxy setting for a given protocol.
**setProxyBypassForLocal** | Specifies a value indicating whether the proxy server should be bypassed if the origin server is on a local network.
**setProxyExceptionList** | Specifies the proxy exception list.
**setProxyName** | Specifies the name of a proxy server to use.
**setProxyPort** | Specifies the proxy port to use.
**setProxySettings** | Specifies the proxy setting for a given protocol.

The **Network** object is accessed through the following property.

**Object** | **Property**
--- | ---
**Player** | **network**

See Also

- **Object Model Reference**

© 2000-2003 Microsoft Corporation. All rights reserved.
Network.bandWidth

The `bandWidth` property retrieves the current bandwidth of the clip.

Syntax

```
player.network.bandWidth
```

Possible Values

This property is a read-only `Number (long)`.

Remarks

This property returns zero if the `Player.URL` property is not set. This property is only valid for streaming media.

Example Code

The following Microsoft JScript example uses `Network.bandWidth` to display the current media bandwidth. The information is displayed in an HTML DIV created with ID = "BW". The Player object was created with ID = "Player".

```
<!-- Create an event handler for play state.-->  
<SCRIPT FOR = Player EVENT = PlayStateChange()>

    switch (Player.playState){
        case 3:
            if (Player.network.bandwidth != 0){
                BW.innerHTML = "Current Bandwidth: " + Player.network.bandWidth;  
                BW.innerHTML += " K bits/second";
            }
            else
                BW.innerHTML = "Bandwidth is only available for streaming media.";
            break;
        default:
    }
</SCRIPT>
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also
Network.bitRate

The **bitRate** property retrieves the current bit rate being received.

**Syntax**

```
player.network.bitRate
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

This value is a combination of the bit rates of both the current video and audio streams.

**Example Code**

The following JScript example uses *Network.bitRate* to display the current media bit rate. The information is displayed in an HTML DIV created with ID = "BR". The Player object was created with ID = "Player".

```javascript
<!--Create an event handler. -->
<SCRIPT FOR = Player EVENT = PlayStateChange()>
    switch (Player.playState){
        case 3:
            if (Player.network.bitRate){
                BR.innerHTML = "Current Bit Rate: " + Player.network.bitRate;
                BR.innerHTML += " K bits/second";
            }
            break;
        default:
    }
</SCRIPT>
```
Network.bufferingCount

The `bufferingCount` property retrieves the number of times buffering occurred during clip playback.

Syntax

`player.network.bufferingCount`

Possible Values

This property is a read-only `Number (long)`.

Remarks

Each time playback is stopped and restarted, this property is set to zero. It is not reset if playback is paused.

Buffering only applies to streaming content. This property returns valid information only during run time when the `Player.URL` property is set.

Example Code

The following JScript example uses `Network.bufferingCount` to display the number of times buffering occurs during playback. The information is displayed in an HTML DIV created with ID = "CB". The Player object was created with ID = "Player".

```html
<!-- Create an event handler. -->
<SCRIPT FOR = Player EVENT = buffering(Start)>
  if (true == Start)
```
Network.bufferingProgress

The **bufferingProgress** property retrieves the percentage of buffering completed.

**Syntax**

```javascript
player.network.bufferingProgress
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

Each time playback is stopped and restarted, this property is set to zero. It is not reset if playback is paused.

Buffering only applies to streaming content. This property returns valid information only during run time, when the **Player.URL** property is set.

Use the **Player.Buffering** event to determine when buffering starts or stops.

**Example Code**

The following JScript example uses **Network.bufferingProgress** to display the percentage of buffering
completed. The information is displayed in an HTML DIV created with ID = "BP". The example uses a timer with a 1-second interval to update the display. The Player object was created with ID = "Player".

<!-- Create an event handler for buffering. -->
<SCRIPT FOR = Player EVENT = buffering(Start)>
    var idI; // Variable for the interval id.

    // Test whether buffering has started or stopped.
    if (true == Start){
        // Start the timer. Call the function to update the display every second.
        idI = window.setInterval("UpdateBP()", 1000);
    }

    else{
        // Buffering is complete. Stop the timer.
        window.clearInterval(idI);
    }
</SCRIPT>

<!-- Put the function to update the display in a separate code block. -->
<SCRIPT>
    function UpdateBP(){
        BP.innerHTML = "Buffering progress: " + Player.network.bufferingProgress;
        BP.innerHTML += percent complete;
    }
</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object
- Player.Buffering Event
- Player.URL

© 2000-2003 Microsoft Corporation. All rights reserved.
The bufferingTime property specifies or retrieves the amount of time in milliseconds allocated for buffering incoming data before playing begins.

**Syntax**

player.network.bufferingTime

**Possible Values**

This property is a read/write **Number (long)** ranging from zero to 60,000 with a default value of 5,000.

**Example Code**

The following JScript example uses `Network.bufferingTime` to specify the number of seconds allocated for buffering incoming data. The information is retrieved from an HTML TEXT INPUT element created with ID = "bufText". The Player object was created with ID = "Player".

```javascript
<!-- Create a BUTTON element to change the bufferingTime value. -->
<INPUT TYPE = "BUTTON" NAME = "bufTime" ID = "bufTime"
    VALUE = "Change Buffer Time"
    onClick = "
        /* Test whether the user entered a valid value. */
        if (bufText.value >= 0 & bufText.value <= 60)
            Player.network.bufferingTime = bufText.value * 1000;
        else
            alert('Buffering time must be between 0 and 60.');"
">

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Network Object**

© 2000-2003 Microsoft Corporation. All rights reserved.
The `downloadProgress` property retrieves the percentage of download completed.

**Syntax**

`player.network.downloadProgress`

**Possible Values**

This property is a read-only `Number (long)`.

**Remarks**

When the Windows Media Player control is connected to a media file that can be played and downloaded at the same time, the `downloadProgress` property returns the percentage of the total file that has been downloaded. This feature is currently supported only on Web servers. The following file formats can be downloaded and played simultaneously:

- Advanced Systems Format (ASF)
- Windows Media Audio (WMA)
- Windows Media Video (WMV)
- MP3
- MPEG
- WAV
- Some AVI files

Use the `Player.Buffering` event to determine when the downloading begins and ends.

**Example Code**

The following JScript example uses `Network.downloadProgress` to display the percentage of downloading completed. The information is displayed in an HTML DIV created with ID = "DP". The example uses a timer with a 1 second interval to update the display. The Player object was created with ID = "Player".

```javascript
<!-- Create an event handler for buffering. -->
<SCRIPT FOR = Player EVENT = buffering(Start)>
  var idI; // Variable for the interval id.

  // Test whether downloading has started or stopped.
  if (true == Start){
    // Start the timer. Call the function to update the display.
    idI = window.setInterval("UpdateDP()", 1000);
  }
  else{
    // Downloading is complete. Stop the timer.
    window.clearInterval(idI);
  }
</SCRIPT>

<!-- Put the function to update the display in a separate code block. -->
<SCRIPT>
function UpdateDP(){
  DP.innerHTML = "";
  DP.innerHTML = "Download progress: " + player.network.downloadProgress;
  DP.innerHTML += " percent complete";
}
</SCRIPT>
```
Network.encodedFrameRate

The `encodedFrameRate` property retrieves the video frame rate specified by the content author in frames per second.

Syntax

```
player.network.encodedFrameRate
```

Possible Values

This property is a read-only `Number (long)`.

Example Code

The following JScript example uses `Network.encodedFrameRate` to display the frame rate specified when the file was encoded. The information is displayed in an HTML DIV created with ID = "FR". The Player object was created with ID = "Player".

```html
<!-- Create an event handler for play state. -->
<SCRIPT FOR = Player EVENT = PlayStateChange(NewState)>
  switch (NewState){
    case 3:
      // Display the encoded frame rate.
      FR.innerHTML = "Encoded Frame Rate: ";
      FR.innerHTML += Player.network.encodedFrameRate;
      break;
```
Network.frameRate

The frameRate property retrieves the current video frame rate in frames per hundred seconds. For example, a value of 2998 indicates 29.98 frames per second.

Syntax

player.network.frameRate

Possible Values

This property is a read-only Number (long).

Example Code

The following JScript example uses Network.frameRate to display the current frame rate. The information is displayed in an HTML DIV created with ID = "FR". The Player object was created with ID = "Player".

<!-- Create an event handler for play state. -->
<SCRIPT FOR = Player EVENT = PlayStateChange(NewState)>
    switch (NewState){
        case 3:
            // Display the current frame rate.
            FR.innerHTML = "Frame Rate: ";
        
</SCRIPT>
FR.innerHTML += Player.network.frameRate;
break;

default:
}
</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object
- Network.encodedFrameRate

Network.framesSkipped

The framesSkipped property retrieves the total number of frames skipped during playback.

Syntax

player.network.framesSkipped

Possible Values

This property is a read-only Number (long).

Example Code

The following JScript example uses Network.framesSkipped to display the total number of frames skipped during playback when the user clicks a button. The information is displayed in an HTML DIV created with ID = "FS". The Player object was created with ID = "Player".

<!-- Create an HTML BUTTON element. -->
<INPUT TYPE = "BUTTON" ID = "skipped" NAME = "skipped"
       VALUE = "Count frames skipped"
Network.getProxyBypassForLocal

The `getProxyBypassForLocal` method retrieves a value indicating whether the proxy server is bypassed if the origin server is on a local network.

Syntax

```javascript
player.network.getProxyBypassForLocal(protocol)
```

Parameters

- `protocol`

String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

Return Values

This method returns a Boolean indicating whether the proxy server is bypassed. The value returned is meaningful only when `getProxySettings` returns a value of two (use manual settings).

Remarks

This method fails unless the calling application is running on the local computer or intranet.

Example Code
The following JScript example uses `Network.getProxyBypassForLocal` to display whether Windows Media Player is set to bypass the proxy server for local addresses. The Player object was created with ID = "Player".

```javascript
// Test whether the HTTP proxy settings are manual.
if (Player.network.getProxySettings("HTTP") == 2)

    // Get the proxy bypass for local value for HTTP.
    var proxyBypassForLocalHTTP = Player.network.getProxyBypassForLocal("HTTP");

// Test whether the MMS proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2)

    // Get the proxy bypass for local value for MMS.
    var proxyBypassForLocalMMS = Player.network.getProxyBypassForLocal("MMS");

// Display the proxy bypass for local values in the browser client area.
// Unavailable proxy bypass for local values will display as "undefined".
document.write("The current HTTP proxy bypass for local value: " + proxyBypassForLocalHTTP);
document.write("<BR>");
document.write("The current MMS proxy bypass for local value: " + proxyBypassForLocalMMS);
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Network Object](#)
- [Network.getProxySettings](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Network.getProxyExceptionList**

The `getProxyExceptionList` method retrieves the proxy exception list.

**Syntax**

```javascript
player.network.getProxyExceptionList(protocol)
```
Parameters

protocol

String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

Return Values

This method returns a String specifying a semicolon-delimited list of hosts for which the proxy server is bypassed. The value returned is meaningful only when getProxySettings returns a value of two (use manual settings).

Remarks

This is a list of computers, domains, and/or addresses that will bypass the proxy server when the host portion of the target URL matches an entry in the list.

The * character can be used as a wildcard for listing entries. For example, *.com would match all hosts in the com domain while 67.* would match all hosts in the 67 class A subnet.

This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following JScript example uses Network.getProxyExceptionList to display the lists of bypassed proxies for the MMS and HTTP protocols. The player object was created with ID = "Player".

```javascript
// Test whether the HTTP proxy settings are manual.
if (Player.network.getProxySettings("HTTP") == 2)

    // Get the proxy exception list for HTTP.
    var proxyExceptionListHTTP = Player.network.getProxyExceptionList("HTTP");

// Test whether the MMS proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2)

    // Get the proxy exception list for MMS.
    var proxyExceptionListMMS = Player.network.getProxyExceptionList("MMS");

// Display the proxy exception lists in the browser client area.
// Unavailable proxy exception lists will display as "undefined".
document.write("The current HTTP proxy exception list: " + proxyExceptionListHTTP);
document.write("<BR>");
document.write("The current MMS proxy exception list: " + proxyExceptionListMMS);
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also
Network.getProxyName

The `getProxyName` method retrieves the name of the proxy server being used.

Syntax

```javascript
player.network.getProxyName(protocol)
```

Parameters

- `protocol` String specifying the protocol name. For a list of supported protocols, see [Supported Protocols and File Types](#).

Return Values

This method returns a String containing the name of the proxy server being used. The value returned is meaningful only when `getProxySettings` returns a value of 2 (use manual settings).

Remarks

This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following JScript example uses `Network.getProxyName` to display the Windows Media Player proxy server names for the HTTP and MMS protocols. The Player object was created with ID = "Player".

```javascript
// Test whether the HTTP proxy settings are manual.
if (Player.network.getProxySettings("HTTP") == 2)

    // Get the proxy server name for HTTP.
    var proxyNameHTTP = Player.network.getProxyName("HTTP");

// Test whether the MMS proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2)

    // Get the proxy server name for MMS.
```
var proxyNameMMS = Player.network.getProxyName("MMS");

// Display the proxy server names in the browser client area.
// Unavailable proxy server names will display as "undefined".
document.write("The current HTTP proxy server name is: " + proxyNameHTTP);
document.write("<BR>");
document.write("The current MMS proxy server name is: " + proxyNameMMS);

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object
- Network.getProxySettings

Network.getProxyPort

The `getProxyPort` method retrieves the proxy port being used.

Syntax

`player.network.getProxyPort(protocol)`

Parameters

- `protocol`
  
  String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

Return Values

This method returns a Number (long) containing the proxy port being used. The value returned is meaningful only when `getProxySettings` returns a value of two (use manual settings).

Remarks
This method fails unless the calling application is running on the local computer or intranet.

**Example Code**

The following JScript example uses `Network.getProxyPort` to display the current Windows Media Player proxy port numbers for the MMS and HTTP protocols. The Player object was created with ID = "Player".

```
// Test whether the HTTP proxy settings are manual.
if (Player.network.getProxySettings("HTTP") == 2)

    // Get the proxy port number for HTTP.
    var proxyPortHTTP = Player.network.getProxyPort("HTTP");

// Test whether the MMS proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2)

    // Get the proxy port number for MMS.
    var proxyPortMMS = Player.network.getProxyPort("MMS");

// Display the port numbers in the browser client area.
// Unavailable port numbers will display as "undefined".
document.write("The current HTTP proxy port is: " + proxyPortHTTP);
document.write("<BR>");
document.write("The current MMS proxy port is: " + proxyPortMMS);
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [Network Object](#)
- [Network.getProxySettings](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Network.getProxySettings**

The `getProxySettings` method retrieves the proxy setting for a given protocol.
Syntax

player.network.getProxySettings(protocol)

Parameters

protocol

String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

Return Values

This method returns a Number (long) containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A proxy server is not being used.</td>
</tr>
<tr>
<td>1</td>
<td>The proxy settings for the current browser are being used (only valid for HTTP).</td>
</tr>
<tr>
<td>2</td>
<td>The manually specified proxy settings are being used.</td>
</tr>
<tr>
<td>3</td>
<td>The proxy settings are being auto-detected.</td>
</tr>
</tbody>
</table>

Remarks

This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following JScript example uses Network.getProxySettings to display a message, which gives information about the current proxy settings of the Player, in the browser window. The Player object was created with ID = "Player".

```javascript
// Retrieve a number representing the current proxy settings.
var proxySetting = Player.network.getProxySettings("MMS");

// Display the message the corresponds to the current settings.
switch(proxySetting){
  case 0:
    document.write("A proxy server is not being used");
    break;
  case 1:
    document.write("The proxy settings for the current browser are being used.");
    break;
  case 2:
    document.write("The manually specified proxy settings are being used.");
    break;
  case 3:
    document.write("The proxy settings are being auto-detected.");
    break;
```
Network.lostPackets

The `lostPackets` property retrieves the number of packets lost.

**Syntax**

```javascript
player.network.lostPackets
```

**Possible Values**

This property is a read-only `Number (long)`.

**Remarks**

This property is only valid for streaming media, and will equal zero when using the HTTP protocol, which is lossless.

Packets may be lost for a number of reasons, such as the type and quality of the network connection.

Each time playback is stopped and restarted, this property is set to zero. It is not reset if playback is paused and restarted. This property returns valid information only during run time, and only if the `Player.URL` property is set.

**Example Code**
The following JScript example uses `Network.lostPackets` to display the total number of packets lost during playback when the user clicks a button. The information is displayed in an HTML DIV created with ID = "LP". The Player object was created with ID = "Player".

```html
<!-- Create an HTML BUTTON element. -->
<INPUT TYPE = "BUTTON" ID = "lostpkts" NAME = "lostpkts"
    VALUE = "Count lost packets"
    onClick = "LP.innerHTML = 'Packets lost: ' + Player.network.lostPackets;">
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Network Object](#)
- [Player.URL](#)

Network.maxBandwidth

The `maxBandwidth` property specifies or retrieves the maximum allowed bandwidth.

**Syntax**

```javascript
player.network.maxBandwidth
```

**Possible Values**

This property is a read/write **Number (long)**.

**Remarks**

This property has no default value. Its value can be specified while the Player is playing, but the change will not take effect until the current media item is released by opening another one or by calling `Player.close`. The Player attempts to achieve the highest bandwidth possible. Only in the case of the value being set will any throttling occur.
This setting is useful for throttling the amount of bandwidth used, particularly in the case of a multiple bit rate (MBR) stream. An MBR stream contains multiple streams with different bit rates. In some cases, it may be desirable to use a stream with a lower bit rate than the client requires. In this case, setting the `maxBandwidth` property will select a lower bit-rate stream. For example, assume that an MBR stream includes streams encoded at 20 kilobits per second (Kbps), 37 Kbps and 200 Kbps. The user's machine has a T1 connection. However, if the desire is to use much less bandwidth than is available, setting the `maxBandwidth` property to 50,000 (50 Kbps) will select the 37 Kbps stream instead of the 200 Kbps stream.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object
- Player.close

Network.maxBitRate

The `maxBitRate` property retrieves the maximum possible video bit rate.

Syntax

```
player.network.maxBitRate
```

Possible Values

This property is a read-only `Number (long)`.

Requirements

Version: Windows Media Player version 7.0 or later.
Network.receivedPackets

The `receivedPackets` property retrieves the number of packets received.

Syntax

```
player.network.receivedPackets
```

Possible Values

This property is a read-only `Number (long)`.

Remarks

Each time clip playback is stopped and restarted, this property is set to zero. It is not reset if file playback is paused.

Example Code

The following JScript example uses `Network.receivedPackets` to display the number of packets received. The information is displayed in an HTML DIV created with ID = "RP". The example uses a timer with a 1-second interval to update the display. The Player object was created with ID = "Player".

```javascript
<!-- Create an event handler for play state change. -->
<SCRIPT FOR = Player EVENT = PlayStateChange(NewState)>

    var idI; // Variable for the interval id.

    // Test whether packets may be arriving.
    switch (NewState){
        case 1, 2, 4, 5, 7, 8, 9:
            window.clearInterval(idI);
            break;
```
Network.receptionQuality

The receptionQuality property retrieves the percentage of packets received in the last 30 seconds.

Syntax

player.network.receptionQuality

Possible Values

This property is a read-only Number (long).

Remarks

The number of packets received, lost, and recovered during streaming is monitored once every second. receptionQuality is the percentage of packets not lost during the last 30 seconds.
Each time file playback is stopped and restarted, this property is set to zero. It is not reset if file playback is paused.

This property returns valid information only during run time and only if the Player.URL property is also set.

**Example Code**

The following JScript example uses `Network.receptionQuality` to display the percentage of packets received. The information is displayed in an HTML DIV created with ID = "RQ". The example uses a timer with a 30-second interval to update the display. The Player object was created with ID = "Player".

```html
<!-- Create an event handler for play state change. -->
<SCRIPT FOR = Player EVENT = PlayStateChange(NewState)>
    var idI; // Variable for the interval id.

    // Test whether content is playing.
    if (3 == NewState){
        // Start the timer. Update the display every 30 seconds.
        idI = window.setInterval("UpdateRQ()", 30000);
    }

    else{
        // Not playing; stop the timer.
        window.clearInterval(idI);
    }
</SCRIPT>

<!-- Put the function to update the display in a separate code block. -->
<SCRIPT>
function UpdateRQ(){
    RQ.innerHTML = "
    RQ.innerHTML = "Reception quality: " + Player.network.receptionQuality;
    RQ.innerHTML += ";
}
</SCRIPT>

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- Network Object
- Player.URL

© 2000-2003 Microsoft Corporation. All rights reserved.
Network.recoveredPackets

The `recoveredPackets` property retrieves the number of recovered packets.

**Syntax**

```
player.network.recoveredPackets
```

**Possible Values**

This property is a read-only `Number` (long).

**Remarks**

Each time file playback is stopped and restarted, this property is set to zero. It is not reset if file playback is paused.

This property returns valid information only during run time and only if the `Player.URL` property is also set. It will equal zero when using the HTTP protocol, which is lossless.

**Example Code**

The following JScript example uses `Network.recoveredPackets` to display the number of recovered packets. The information is displayed in an HTML DIV created with ID = "PR". The example uses a timer with a 1-second interval to update the display. The Player object was created with ID = "Player".

```html
<!-- Create an event handler for play state change. -->
<script for = Player event = PlayStateChange(newState)>

var idI; // Variable for the interval id.

// Test whether content is playing.
if (3 == newState){
    // Start the timer. Call the function to update the display every second.
    idI = window.setInterval("UpdatePR()", 1000);
}
else{
    // Not playing; stop the timer.
    window.clearInterval(idI);
}
</script>

<!-- Put the function to update the display in a separate code block. -->
<script>

function UpdatePR()
{
    PR.innerHTML = "";
    PR.innerHTML = "Packets recovered: " + Player.network.recoveredPackets;
}
</script>
```
Network.setProxyBypassForLocal

The `setProxyBypassForLocal` method specifies a value indicating whether the proxy server is bypassed if the origin server is on a local network.

Syntax

```
player.network.setProxyBypassForLocal(protocol, bypass)
```

Parameters

- **protocol**
  
  String specifying the protocol name. For a list of supported protocols, see [Supported Protocols and File Types](#).

- **bypass**
  
  Boolean specifying whether the proxy server is bypassed.

Return Values

This method does not return a value.

Remarks

This method has no effect unless `getProxySettings` returns a value of 2 (use manual settings).
This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following JScript example uses `Network.setProxyBypassForLocal` to specify whether the Windows Media Player proxy server is bypassed, when using the MMS protocol, if the origin server is on a local network. The Player object was created with ID = "Player".

```javascript
// Test whether the proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2) {
    // Prompt the user for a setting. Store the response in a variable.
    var proxybypass = confirm("Bypass proxy server for local addresses? \n    OK = Yes \n    Canc

    // Set the proxy bypass value using the user input.
    Player.network.setProxyBypassForLocal("MMS", proxybypass);
}
else
    // Warn that proxy settings must be set to 2.
    alert("Proxy settings must be manual!");
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Network Object](#)

---

Network.setProxyExceptionList

The `setProxyExceptionList` method specifies the proxy exception list.

**Syntax**

```javascript
player.network.setProxyExceptionList(protocol, list)
```
Parameters

protocol

String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

list

String specifying a semicolon-delimited list of hosts for which the proxy server is bypassed. Leading and trailing spaces and semicolons should not be present.

Return Values

This method does not return a value.

Remarks

This is a list of computers, domains, and/or addresses that will bypass the proxy server when the host portion of the target URL matches an entry in the list.

The * character can be used as a wildcard for listing entries. For example, *.com would match all hosts in the com domain, while 67.* would match all hosts in the 67 class A subnet.

This method has no effect unless getProxySettings returns a value of 2 (use manual settings).

This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following JScript example uses Network.setProxyExceptionList to specify a list of hosts for which the proxy server is bypassed when using the MMS protocol. The new list is retrieved from an HTML TEXT element with ID = "XLIST". The Player object was created with ID = "Player".

```javascript
// Test whether proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2){

    // Store the user's new exception list.
    var proxyxlist = XLIST.value;

    // Set the exception list.
    Player.network.setProxyExceptionList("MMS", proxyxlist);
}
else

    // Warn that the proxy settings must be set to 2.
    alert("Proxy settings must be manual!");
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.
The `setProxyName` method specifies the name of the proxy server to use.

**Syntax**

```
player.network.setProxyName(protocol, name)
```

**Parameters**

- `protocol`
  
  *String* specifying the protocol name. For a list of supported protocols, see [Supported Protocols and File Types](#).

- `name`
  
  *String* specifying the name of the proxy server to use.

**Return Values**

This method does not return a value.

**Remarks**

This method has no effect unless `getProxySettings` returns a value of 2 (use manual settings).

This method fails unless the calling application is running on the local computer or intranet.

**Example Code**

The following JScript example uses `Network.setProxyName` to specify the name of the Windows Media Player proxy server for the MMS protocol. The new name is retrieved from an HTML TEXT element with ID = "NAME". The Player object was created with ID = "Player".
// Test whether proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2){

    // Store the new proxy server name specified by the user.
    var proxynname = NAME.value;

    // Set the proxy server name.
    Player.network.setProxyName("MMS", proxynname);
}
else

    // Warn that proxy settings must be set to 2.
    alert("Proxy settings must be manual!");

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object

Network.setProxyPort

The setProxyPort method specifies the proxy port to use.

Syntax

player.network.setProxyPort(protocol, port)

Parameters

protocol

String specifying the protocol name. For a list of supported protocols, see Supported Protocols and File Types.

port
**Number (long)** specifying the proxy port to use.

**Return Values**

This method does not return a value.

**Remarks**

This method has no effect unless **getProxySettings** returns a value of 2 (use manual settings).

This method fails unless the calling application is running on the local computer or intranet.

**Example Code**

The following JScript example uses **Network.setProxyPort** to specify the Windows Media Player proxy port number for the MMS protocol. The port number is retrieved from an HTML INPUT element with ID = "PORT". The Player object was created with ID = "Player".

```javascript
// Test whether proxy settings are manual.
if (Player.network.getProxySettings("MMS") == 2){
    // Store the port number specified by the user.
    var portnumber = PORT.value;

    // Set the port number for the MMS protocol.
    Player.network.setProxyPort("MMS", portnumber);
}
else
    // Warn that proxy settings must be set to 2.
    alert("Proxy settings must be manual!");
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Network Object**
Network.setProxySettings

The `setProxySettings` method specifies the proxy setting for a given protocol.

Syntax

```
player.network.setProxySettings(protocol, setting)
```

Parameters

- **protocol**
  
  *String* specifying the protocol name. For a list of supported protocols, see [Supported Protocols and File Types](#).

- **setting**
  
  *Number (long)* containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use a proxy server.</td>
</tr>
<tr>
<td>1</td>
<td>Use the proxy settings of the current browser (only valid for HTTP).</td>
</tr>
<tr>
<td>2</td>
<td>Use the manually specified proxy settings.</td>
</tr>
<tr>
<td>3</td>
<td>Auto-detect the proxy settings.</td>
</tr>
</tbody>
</table>

Return Values

This method does not return a value.

Remarks

This method fails unless the calling application is running on the local computer or intranet.

Example Code

The following example uses JScript with an HTML SELECT element to allow the user to specify the Windows Media Player proxy setting for the HTTP protocol. The Player object was created with ID = "Player".

```html
<SELECT ID = HTTPsetproxy  NAME = "HTTPsetproxy"  LANGUAGE="JScript"
  onChange = " /* Store the current selection. */
    var setting = this.value;
  /* Change the proxy setting. */
```
Player.network.setProxySettings('HTTP', setting);
">

<OPTION VALUE=0>Do not use a proxy server
<OPTION VALUE=1>Use the browser settings
<OPTION VALUE=2>Use manual settings
<OPTION VALUE=3>Auto-detect settings

</SELECT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

• Network Object

Network.sourceProtocol

The sourceProtocol property retrieves the source protocol used to receive data.

Syntax

player.network.sourceProtocol

Possible Values

This property is a read-only String.

Remarks

This property is set to "" (empty string) when playing media from a CD or DVD.

Example Code

The following JScript example uses Network.sourceProtocol to display the source protocol used to receive
data. The information is displayed in an HTML DIV created with ID = "SP". The Player object was created with ID = "Player".

<!-- Create an event handler for play state change. -->
<SCRIPT FOR = Player EVENT = PlayStateChange(NewState)>
    if (3 == NewState){
        SP.innerHTML = "Source protocol: " + Player.network.sourceProtocol;
    }
</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Network Object

Player Object

The Player object is the root object for the Windows Media Player control. It supports the properties, methods, and events listed in the following tables.

The Player object supports the following properties. Properties marked with an asterisk (*) are not accessible to skins.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdromCollection</td>
<td>Retrieves the CdromCollection object.</td>
</tr>
<tr>
<td>closedCaption</td>
<td>Retrieves the ClosedCaption object.</td>
</tr>
<tr>
<td>controls</td>
<td>Retrieves the Controls object.</td>
</tr>
<tr>
<td>currentMedia</td>
<td>Specifies or retrieves the current Media object.</td>
</tr>
<tr>
<td>currentPlaylist</td>
<td>Specifies or retrieves the current Playlist object.</td>
</tr>
</tbody>
</table>
dvd
  Retrieves the DVD object.

enableContextMenu *
  Specifies or retrieves a value indicating whether to enable the context menu, which appears when the right mouse button is clicked.

enabled *
  Specifies or retrieves a value indicating whether the Windows Media Player control is enabled.

error
  Retrieves the Error object.

fullScreen *
  Specifies or retrieves a value indicating whether video content is played back in full-screen mode.

isOnline
  Retrieves a value indicating whether the user is connected to a network.

isRemote *
  Retrieves a value indicating whether the Windows Media Player control is running in remote mode.

mediaCollection
  Retrieves the MediaCollection object.

network
  Retrieves the Network object.

openState
  Retrieves a value indicating the state of the content source.

playerApplication *
  Retrieves the PlayerApplication object when a remoted Windows Media Player control is running.

playlistCollection
  Retrieves the PlaylistCollection object.

playState
  Retrieves a value indicating the state of the Windows Media Player operation.

settings
  Retrieves the Settings object.

status
  Retrieves a value indicating the current status of Windows Media Player.

stretchToFit *
  Specifies or retrieves a value indicating whether video will stretch to fit size of the Windows Media Player control video display.

uiMode *
  Specifies or retrieves a value indicating which controls are shown in the user interface when Windows Media Player is embedded in a Web page.

URL
  Specifies or retrieves the name of the clip to play.

versionInfo
  Retrieves a String value specifying the version of the Windows Media Player.

windowlessVideo *
  Specifies or retrieves a value indicating whether the Windows Media Player control renders video in windowless mode.

* Not accessible to skins.
The **Player** object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>close</strong></td>
<td>Releases Windows Media Player resources.</td>
</tr>
<tr>
<td><strong>launchURL</strong></td>
<td>Sends a URL to the user's default browser to be rendered.</td>
</tr>
<tr>
<td><strong>newMedia</strong></td>
<td>Creates a new <strong>Media</strong> object.</td>
</tr>
<tr>
<td><strong>newPlaylist</strong></td>
<td>Creates a new <strong>Playlist</strong> object.</td>
</tr>
<tr>
<td><strong>openPlayer</strong></td>
<td>Opens Windows Media Player using the specified URL.</td>
</tr>
</tbody>
</table>

The **Player** object supports the following events. Events marked with an asterisk (*) are not accessible to skins. For information about handling mouse and keyboard events in skins, see [External Events](#).

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AudioLanguageChange</strong></td>
<td>Occurs when the current audio language changes.</td>
</tr>
<tr>
<td><strong>Buffering</strong></td>
<td>Occurs when the Windows Media Player control begins or ends buffering.</td>
</tr>
<tr>
<td><strong>CdromMediaChange</strong></td>
<td>Occurs when a CD or DVD is inserted into or ejected from a CD-ROM or DVD-ROM drive.</td>
</tr>
<tr>
<td><strong>Click</strong></td>
<td>Occurs when the user clicks a mouse button.</td>
</tr>
<tr>
<td><strong>CurrentItemChange</strong></td>
<td>Occurs when <strong>Controls.currentItem</strong> changes.</td>
</tr>
<tr>
<td><strong>CurrentMediaItemAvailable</strong></td>
<td>Occurs when a graphic metadata item in the current media item becomes available.</td>
</tr>
<tr>
<td><strong>CurrentPlaylistChange</strong></td>
<td>Occurs when something changes within the current playlist.</td>
</tr>
<tr>
<td><strong>CurrentPlaylistItemAvailable</strong></td>
<td>Occurs when the current playlist item becomes available.</td>
</tr>
<tr>
<td><strong>Disconnect</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>DomainChange</strong></td>
<td>Occurs when the DVD domain changes.</td>
</tr>
<tr>
<td><strong>DoubleClick</strong></td>
<td>Occurs when the user double-clicks a mouse button.</td>
</tr>
<tr>
<td><strong>DurationUnitChange</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>EndOfStream</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td>Occurs when the Windows Media Player control has an error condition.</td>
</tr>
<tr>
<td><strong>KeyDown</strong></td>
<td>Occurs when a key is pressed.</td>
</tr>
<tr>
<td><strong>KeyPress</strong></td>
<td>Occurs when a key is pressed and then released.</td>
</tr>
</tbody>
</table>
**KeyUp**  * Occurs when a key is released.

**MarkerHit**
Occurs when a marker is reached.

**MediaChange**
Occurs when a media item changes.

**MediaCollectionAttributeStringAdded**
Occurs when an attribute value is added to **Media Library**.

**MediaCollectionAttributeStringChanged**
Occurs when an attribute value in **Media Library** is changed.

**MediaCollectionAttributeStringRemoved**
Occurs when an attribute value is removed from **Media Library**.

**MediaCollectionChange**
Occurs when the media collection changes.

**MediaError**
Occurs when the **Media** object has an error condition.

**ModeChange**
Occurs when a mode of the player is changed.

**MouseDown**  *
Occurs when a mouse button is pressed.

**MouseMove**  *
Occurs when the mouse pointer is moved.

**MouseUp**  *
Occurs when a mouse button is released.

**NewStream**
Reserved for future use.

**OpenPlaylistSwitch**
Occurs when a title on a DVD begins playing.

**OpenStateChange**
Occurs when the Windows Media Player control changes state.

**PlayerDockedStateChange**
Occurs when a remoted Windows Media Player control docks or undocks.

**PlayerReconnect**
Occurs when a remoted Windows Media Player control reconnects to Windows Media Player.

**PlaylistChange**
Occurs when a playlist changes.

**PlaylistCollectionChange**
Occurs when something changes in the playlist collection.

**PlaylistCollectionPlaylistAdded**
Occurs when a playlist is added to the playlist collection.

**PlaylistCollectionPlaylistRemoved**
Occurs when a playlist is removed from the playlist collection.

**PlaylistCollectionPlaylistSetAsDeleted**
Reserved for future use.

**PlayStateChange**
Occurs when the play state of the Windows Media Player control changes.

**PositionChange**
Occurs when the current position of the media item has been changed.

**ScriptCommand**
Occurs when a synchronized command or URL is
* Not accessible to skins. For information about handling mouse and keyboard events in skins, see Ambient Event Handlers.

When embedded in a Web page, the Player object can be accessed by using the ID value specified in the OBJECT tag. Within a skin definition file, it is accessed by using the player global attribute. For illustration purposes, player will be used as the object ID in the reference syntax sections.

See Also

- Object Model Reference

---

**Player.AudioLanguageChange Event**

The AudioLanguageChange event occurs when the current audio language changes.

**Syntax**

`AudioLanguageChange(LangID)`

**Parameters**

- `LangID`  
  Number (long) specifying the new locale identifier (LCID).

**Remarks**

An LCID uniquely identifies a particular language dialect, called a locale.
The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported Microsoft JScript file by using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Controls.currentAudioLanguage](#)
- [Player Object](#)

---

**Player.Buffering Event**

The **Buffering** event occurs when the Windows Media Player control begins or ends buffering or downloading.

**Syntax**

Buffering(Start)

**Parameters**

- **Start**

  *Boolean* containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Buffering has started.</td>
</tr>
<tr>
<td>false</td>
<td>Buffering has ended.</td>
</tr>
</tbody>
</table>

**Remarks**
Use this event to determine when buffering or downloading starts or stops. You can use the same event block for both cases and test `Network.bufferingProgress` and `Network.downloadProgress` to determine whether Windows Media Player is currently buffering or downloading content.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file by using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- `Network.bufferingProgress`
- `Network.downloadProgress`
- `Player Object`

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.cdromCollection**

The `cdromCollection` property retrieves the `CdromCollection` object.

**Syntax**

```
player.cdromCollection
```

**Possible Values**

This property is a read-only `CdromCollection` object.

**Remarks**

To retrieve the value of this property, read access to `Media Library` is required. For more information, see `Media Library Access`. 
Player.CdromMediaChange Event

The **CdromMediaChange** event occurs when a CD or DVD is inserted into or ejected from a CD-ROM or DVD-ROM drive.

**Syntax**

CdromMediaChange(CdromNum)

**Parameters**

*CdromNum*

*Number (long)* specifying the index of the CD-ROM drive.

**Remarks**

The index of the CD-ROM drive corresponds to the index of a **Cdrom** object in the **CdromCollection**.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**
Player.Click Event

The Click event occurs when the user clicks a mouse button.

Syntax

Click(nButton, nShiftState, fX, fY)

Parameters

nButton

Number (int) specifying a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

nShiftState

Number (int) specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

fX

Number (long) specifying the x coordinate of the mouse pointer relative to the upper left-hand corner of the control.
**Number** *(long)* specifying the *y* coordinate of the mouse pointer relative to the upper left-hand corner of the control.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Player Object**

---

**Player.close**

The **close** method releases Windows Media Player resources.

**Syntax**

`player.close()`

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.
Remarks

This method closes the current digital media file, not the Player itself.

Example Code

The following example creates an HTML BUTTON element that, when clicked, stops playback in Windows Media Player and releases the resources in use. The Player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON" ID = "CLOSEIT" VALUE = "Close it" onClick = "
    /* Close the Player object. */
    Player.close();"
">
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object

Player.closedCaption

The closedCaption property retrieves the ClosedCaption object.

Syntax

```javascript
player.closedCaption
```

Possible Values

This property is a read-only ClosedCaption object.

Requirements
Player.controls

The controls property retrieves the Controls object.

Syntax

player.controls

Possible Values

This property is a read-only Controls object.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls Object
- Player Object
Player.CurrentItemChange Event

The CurrentItemChange event occurs when Controls.currentItem changes.

Syntax

CurrentItemChange()

Example Code

The following JScript example demonstrates an event handler for the Player.currentItemChange event. The player object was created with ID = "Player".

<!-- Create an HTML text box to display the media name. -->
<INPUT TYPE="TEXT" NAME="MEDIA">

<!-- Create an event handler. -->
<SCRIPT LANGUAGE = "JScript"  FOR = Player EVENT = currentItemChange()>

    // Display the name of the new media item.
    MEDIA.value = Player.currentMedia.name;

</SCRIPT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls.currentItem
- Player Object
Player.currentMedia

The `currentMedia` property specifies or retrieves the current `Media` object.

Syntax

```
player.currentMedia
```

Possible Values

This property is a read/write `Media` object.

Remarks

If the `Settings.autoStart` property is true, file playback begins automatically whenever you set `currentMedia`.

This property takes a `Media` object, which can be acquired by using `Playlist.item`. To load a `Media` item using a file name, set the `URL` property or use `newMedia`.

Example Code

The following JScript example retrieves the first media item in `Media Library`. It then uses `currentMedia` to make the retrieved media item the current media item, and then to display the name of the current media item. The Player object was created with ID = "Player".

```jscript
// Retrieve the first media item from Media Library
var firstMedia = Player.mediaCollection.getAll().item(0);

// Make the retrieved media item the current media item.
Player.currentMedia = firstMedia;

// Display the name of the current media item.
document.write("Found first media item. Name = " + Player.currentMedia.name);
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- `Media Object`
- `Player Object`
- `Player.newMedia`
Player.CurrentMediaItemAvailable Event

The **CurrentMediaItemAvailable** event occurs when a graphic metadata item in the current media item becomes available.

**Syntax**

```javascript
CurrentMediaItemAvailable(bstrItemName)
```

**Parameters**

*bstrItemName*

*String* containing the name of the current media item.

**Remarks**

Because playback can begin before a media item is fully downloaded, any metadata graphics contained in the media item (such as album cover art) may not be available when it starts to play. This event alerts you when a metadata graphic item is finished downloading. You can then retrieve the **Media** object by passing the value of *bstrItemName* to the **MediaCollection.getByName** method, after which you can access the metadata graphic item by using **Media.getItemInfoByType** and specifying **WM/Picture** for the attribute name.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**
Player.currentPlaylist

The currentPlaylist property specifies or retrieves the current Playlist object.

Syntax

player.currentPlaylist

Possible Values

This property is a read/write Playlist object.

Remarks

If the Settings.autoStart property is true, file playback begins automatically whenever you set currentPlaylist.

This property takes a Playlist object, which can be acquired in several ways, such as by calling PlaylistArray.item or PlaylistCollection.newPlaylist. To load a Playlist item using a file name, set the URL property or use Player.newPlaylist.

Example Code

The following JScript example retrieves the first playlist in Media Library. It then uses currentPlaylist to make the retrieved playlist the current playlist, and then to display the name of the current playlist. The Player object was created with ID = "Player".

```
// Retrieve the first playlist from Media Library.
var firstPL = Player.playlistCollection.getAll().item(0);

// Make the retrieved playlist the current playlist.
Player.currentPlaylist = firstPL;

// Display the name of the current playlist.
document.write("Found first playlist. Name: " + Player.currentPlaylist.name);
```

Requirements
Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.currentPlaylist
- Player.newPlaylist
- Playlist Object
- PlaylistArray.item
- PlaylistCollection.newPlaylist
- Settings.autoStart

Player.CurrentPlaylistChange Event

The CurrentPlaylistChange event occurs when something changes within the current playlist.

Syntax

CurrentPlaylistChange(change)

Parameters

change

Number (long) indicating what type of change occurred to the playlist. See the Player.PlaylistChange event for a table of possible values.

Remarks

This event does not occur when a different playlist becomes the current playlist. It only occurs when a change happens within the current playlist, such as a media item being appended to the playlist.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.
Example Code

The following JScript example updates the text in an HTML DIV element, named PlItems, to display the names of the media items in the current playlist. The Player object was created with ID = "Player".

```html
<!-- Create an event handler for current playlist change. -->
<SCRIPT FOR = "Player" EVENT = "currentPlaylistChange(change)">
    switch (change){
        // Only update for move, delete, insert, and append events.
        case 3, 4, 5, 6:
            // Clear the contents of the DIV.
            PlItems.innerHTML = "";
            // Loop through the playlist and display each item name.
            for (var i = 0; i < Player.currentPlaylist.count; i++){
                PlItems.innerHTML += Player.currentPlaylist.item(i).name;
                PlItems.innerHTML += "<br>";
            }
            break;
        default:
    }
</SCRIPT>
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.currentPlaylist
- Player.PlaylistChange

© 2000-2003 Microsoft Corporation. All rights reserved.

Player.CurrentPlaylistItemAvailable Event

The CurrentPlaylistItemAvailable event occurs when the current playlist becomes available.
Syntax

CurrentPlaylistItemAvailable(bstrItemName)

Parameters

bstrItemName

String containing the name of the current playlist item.

Remarks

The name of the current playlist can be used to retrieve the corresponding Playlist object using the PlaylistCollection.getByName method.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Playlist Object
- PlaylistCollection.getByName

© 2000-2003 Microsoft Corporation. All rights reserved.

Player.DomainChange Event

The DomainChange event occurs when the DVD domain changes.

Syntax

DomainChange(strDomain)
Parameters

strDomain

String indicating the new domain. Contains one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstPlay</td>
<td>Performing default initialization of a DVD disc.</td>
</tr>
<tr>
<td>videoManagerMenu</td>
<td>Displaying menus for whole disc. Also known as Root Menu or topMenu.</td>
</tr>
<tr>
<td>videoTitleSetMenu</td>
<td>Displaying menus for current title set. Also known as titleMenu</td>
</tr>
<tr>
<td>title</td>
<td>Displaying the current title.</td>
</tr>
<tr>
<td>stop</td>
<td>The DVD Navigator is in the DVD Stop domain.</td>
</tr>
</tbody>
</table>

Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

Platform: Windows XP or later.

See Also

- DVD Object
- Player Object

© 2000-2003 Microsoft Corporation. All rights reserved.
Player.DoubleClick Event

The **DoubleClick** event occurs when the user double-clicks a mouse button.

**Syntax**

```plaintext
DoubleClick(nButton, nShiftState, fX, fY)
```

**Parameters**

`nButton`

**Number (int)** specifying a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

`nShiftState`

**Number (int)** specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

`fX`

**Number (long)** specifying the x coordinate of the mouse pointer relative to the upper left-hand corner of the control.

`fY`

**Number (long)** specifying the y coordinate of the mouse pointer relative to the upper left-hand corner of the control.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Player Object**
The `dvd` property retrieves the **DVD** object.

**Syntax**

```plaintext
player.dvd
```

**Possible Values**

This property is a read-only **DVD** object.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**Platform:** Windows XP or later.

**See Also**

- **DVD Object**
- **Player Object**
Player.enableContextMenu

The `enableContextMenu` property specifies or retrieves a value indicating whether to enable the context menu, which appears when the right mouse button is clicked.

Syntax

`player.enableContextMenu`

Possible Values

This property is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Enable the context menu.</td>
</tr>
<tr>
<td>false</td>
<td>Do not enable the context menu.</td>
</tr>
</tbody>
</table>

Remarks

During full-screen playback, Windows Media Player hides the mouse cursor when `enableContextMenu` equals `false` and `uiMode` equals "none".

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
The **enabled** property specifies or retrieves a value indicating whether the Windows Media Player control is enabled.

**Syntax**

```
player.enabled
```

**Possible Values**

This property is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The Windows Media Player control is enabled.</td>
</tr>
<tr>
<td>false</td>
<td>The Windows Media Player control is disabled.</td>
</tr>
</tbody>
</table>

**Remarks**

If `enabled` equals false then during full-screen playback Windows Media Player hides the user controls.

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [Player Object](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

The **error** property retrieves the **Error** object.

**Syntax**

```
player.error

Possible Values

This property is a read-only Error object.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Error Object
- Player Object

Player.Error Event

The Error event occurs when there is an error condition.

Syntax

Error()

Example Code

The following JScript example creates an event handler to display the description text for the first error in the error queue. The Player object was created with ID = "Player".

<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = error()>

  // Get the description of the first error.
  var errDesc = Player.error.item(0).errorDescription;

  // Display the error description.
  alert(errDesc);

</SCRIPT>
Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- ErrorItem.errorDescription
- Error.item
- Player Object

Player.fullScreen

The fullScreen property specifies or retrieves a value indicating whether video content is played back in full-screen mode.

Syntax

player.fullScreen

Possible Values

This property is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Video content is played back in full-screen mode.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Video content is not played back in full-screen mode.</td>
</tr>
</tbody>
</table>

Remarks

For full-screen mode to work properly when embedding the Windows Media Player control, the video display area must have a height and width of at least one pixel. If uiMode is set to "mini" or "full", the height of the control itself must be 65 or greater to accommodate the video display area in addition to the user interface.
If `uiMode` is set to "invisible", then setting this property to true raises an error and does not affect the behavior of the control.

During full-screen playback, Windows Media Player hides the mouse cursor when `enableContextMenu` equals false and `uiMode` equals "none".

If `uiMode` is set to "full" or "mini", the Player displays transport controls in full-screen mode when the mouse cursor moves. After a brief interval of no mouse movement, the transport controls are hidden. If `uiMode` is set to "none", no controls are displayed in full-screen mode.

**Note**  Displaying transport controls in full-screen mode requires the Windows XP operating system.

If transport controls are not displayed in full-screen mode, then Windows Media Player automatically exits full-screen mode when playback stops.

**Example Code**

The following example creates an HTML input button that uses `Player.fullScreen` to switch an embedded player object to full-screen mode. The player object was created with ID = "Player".

```html
<INPUT type = button
value = "Full Screen"
name = FSBUTTON
onclick = "
  /* Check to be sure the player is playing. */
  if (Player.playState == 3)
    Player.fullScreen = 'true';
"/>
```

**Note** Always be sure to inform the user how to return from full-screen mode.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player Object](#)
Player.isOnline

The isOnline property retrieves a value indicating whether the user is connected to a network.

Syntax

player.isOnline

Possible Values

This property is a read-only Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The user is connected to a network.</td>
</tr>
<tr>
<td>false</td>
<td>The user is not connected to a network.</td>
</tr>
</tbody>
</table>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object

Player.isRemote

The isRemote property retrieves a value indicating whether the Windows Media Player control is running in remote mode.

Syntax
player.isRemote

Possible Values

This property is a read-only Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The Player control is running in remote mode.</td>
</tr>
<tr>
<td>false</td>
<td>The Player control is running in local mode.</td>
</tr>
</tbody>
</table>

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Remoting the Windows Media Player Control

Player.KeyDown Event

The **KeyDown** event occurs when a key is pressed.

Syntax

```plaintext
KeyDown(nKeyCode, nShiftState)
```

Parameters

`nKeyCode`

Number (int) specifying which physical key is pressed. For possible values, see the Remarks section.
**nShiftState**

**Number (int)** specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

**Remarks**

The nKeyCode argument specifies a physical key. The following tables show the possible values for the major keys on a standard keyboard.

Values for the main keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Z</td>
<td>65-90</td>
</tr>
<tr>
<td>0-9</td>
<td>48-56</td>
</tr>
<tr>
<td>F1-F12</td>
<td>112-123</td>
</tr>
<tr>
<td>ESC</td>
<td>27</td>
</tr>
<tr>
<td>TAB</td>
<td>9</td>
</tr>
<tr>
<td>CAPS LOCK</td>
<td>20</td>
</tr>
<tr>
<td>SHIFT (left or right)</td>
<td>16</td>
</tr>
<tr>
<td>CTRL (left or right)</td>
<td>17</td>
</tr>
<tr>
<td>ALT (left or right)</td>
<td>18</td>
</tr>
<tr>
<td>SPACE</td>
<td>32</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>8</td>
</tr>
<tr>
<td>ENTER</td>
<td>13</td>
</tr>
<tr>
<td>Windows logo key, left</td>
<td>91</td>
</tr>
<tr>
<td>Windows logo key, right</td>
<td>92</td>
</tr>
<tr>
<td>Application key</td>
<td>93</td>
</tr>
</tbody>
</table>

Values for the number pad keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>96-105</td>
</tr>
<tr>
<td>NUM LOCK</td>
<td>144</td>
</tr>
<tr>
<td>DIVIDE (/)</td>
<td>111</td>
</tr>
<tr>
<td>MULTIPLY (*)</td>
<td>106</td>
</tr>
</tbody>
</table>
Values for the navigation keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>45</td>
</tr>
<tr>
<td>DELETE</td>
<td>46</td>
</tr>
<tr>
<td>HOME</td>
<td>36</td>
</tr>
<tr>
<td>END</td>
<td>35</td>
</tr>
<tr>
<td>PAGE UP</td>
<td>33</td>
</tr>
<tr>
<td>PAGE DOWN</td>
<td>34</td>
</tr>
<tr>
<td>UP ARROW</td>
<td>38</td>
</tr>
<tr>
<td>DOWN ARROW</td>
<td>40</td>
</tr>
<tr>
<td>LEFT ARROW</td>
<td>37</td>
</tr>
<tr>
<td>RIGHT ARROW</td>
<td>39</td>
</tr>
</tbody>
</table>

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
Player.KeyPress Event

The **KeyPress** event occurs when a key is pressed and released.

Syntax

`KeyPress(nKeyAscii)`

Parameter

`nKeyAscii`

**Number (int)** specifying the standard numeric ANSI code for the character.

Remarks

This event occurs when the keystroke results in any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- **Player Object**
Player.KeyUp Event

The KeyUp event occurs when a key is released.

Syntax

KeyUp(nKeyCode, nShiftState)

Parameters

nKeyCode

Number (int) specifying which physical key is pressed. For possible values, see Player.KeyDown Event.

nShiftState

Number (int) specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
Player.launchURL

The `launchURL` method sends a URL to the user's default browser to be rendered.

**Syntax**

```javascript
player.launchURL(URL)
```

**Parameters**

- `URL`  
  String value representing the URL to launch.

**Return Values**

This method does not return a value.

**Remarks**

This method only opens Web pages using the HTTP or HTTPS protocols.

**Example Code**

The following example creates an HTML BUTTON element that, when clicked, displays the Microsoft Web site in a new browser window. The Player element was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON" ID = "GOTOMS" VALUE = "Microsoft.com" onClick = " /* Open the Microsoft Web site. */
    Player.launchURL('http://www.microsoft.com');">
```

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- [Player Object](#)
Player.MarkerHit Event

The MarkerHit event occurs when a marker is reached.

Syntax

MarkerHit(MarkerNum)

Parameters

MarkerNum

Number (long) indicating the number of the marker that was hit.

Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
Player.MediaChange Event

The **MediaChange** event occurs when a media item changes.

Syntax

```
MediaChange(Item)
```

Parameters

*Item*

**Media** object representing the item that changed.

Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Example Code

The following JScript example uses an HTML DIV element, named MediaName, to display the name of the current media item. The code updates the text in the DIV with each occurrence of the **mediaChange** event. The Player object was created with ID = "Player".

```html
<!-- Create an event handler for media change. -->
<SCRIPT FOR = "Player" EVENT = "mediaChange(Item)">
    // Test whether a valid currentMedia object exists.
    if (Player.currentMedia){
        // Display the name of the current media item.
        MediaName.innerHTML = Player.currentMedia.name;
    }
</SCRIPT>
```

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- **Player Object**
Player.mediaCollection

The `mediaCollection` property retrieves the `MediaCollection` object.

Syntax

```
player.mediaCollection
```

Possible Values

This property is a read-only `MediaCollection` object.

Remarks

To retrieve the value of this property, read access to `Media Library` is required. For more information, see `Media Library Access`.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- `MediaCollection Object`
- `Player Object`
- `Settings.mediaAccessRights`
- `Settings.requestMediaAccessRights`
Player.MediaCollectionAttributeStringAdded Event

The MediaCollectionAttributeStringAdded event occurs when an attribute value is added to Media Library.

Syntax

MediaCollectionAttributeStringAdded(bstrAttribName, bstrAttribVal)

Parameters

bstrAttribName

String specifying the name of the attribute. For information about the attributes supported by Windows Media Player, see Available Attributes.

bstrAttribVal

String specifying the value of the attribute.

Remarks

When a media item is added to Media Library its metadata is added to the MediaCollection object and this event is fired for each attribute added.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection Object
- Player Object
- Player.mediaCollection
Player.MediaCollectionAttributeStringChanged Event

The MediaCollectionAttributeStringChanged event occurs when an attribute value in Media Library is changed.

Syntax

MediaCollectionAttributeStringChanged(bstrAttribName, bstrOldAttribVal, bstrNewAttribVal)

Parameters

bstrAttribName

String specifying the name of the attribute. For information about the attributes supported by Windows Media Player, see Available Attributes.

bstrOldAttribVal

String specifying the old value of the attribute.

bstrNewAttribVal

String specifying the new value of the attribute.

Remarks

When a user modifies Media Library metadata, the MediaCollection object is updated and this event fires for each attribute changed.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.
Player.MediaCollectionAttributeStringRemoved Event

The MediaCollectionAttributeStringRemoved event occurs when an attribute value is removed from Media Library.

Syntax

MediaCollectionAttributeStringRemoved(bstrAttribName, bstrAttribVal)

Parameters

bstrAttribName

String specifying the name of the attribute. For information about the attributes supported by Windows Media Player, see Available Attributes.

bstrAttribVal

String specifying the value of the attribute.

Remarks

When a media item is removed from Media Library its metadata is removed from the MediaCollection object and this event is fired for each attribute removed.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player version 7.0 or later.
Player.MediaCollectionChange Event

The MediaCollectionChange event occurs when the media collection changes.

Syntax

MediaCollectionChange()

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection Object
- Player Object
- Player.mediaCollection
Player.MediaError Event

The MediaError event occurs when the Media object has an error condition.

Syntax

MediaError(pMediaObject)

Parameters

pMediaObject

Media object that has an error condition.

Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player for Windows XP or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
Player.ModeChange Event

The **ModeChange** event occurs when a mode of the player is changed.

**Syntax**

```
ModeChange(ModeName, NewValue)
```

**Parameters**

- **ModeName**
  - String indicating the mode that was changed. Contains one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shuffle</td>
<td>Tracks are played in random order.</td>
</tr>
<tr>
<td>loop</td>
<td>The entire sequence of tracks is played repeatedly.</td>
</tr>
</tbody>
</table>

- **NewValue**
  - Boolean indicating the new state of the specified mode.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

- **Version:** Windows Media Player version 7.0 or later.
- **Header:** Defined in wmp.idl; include wmp.h.
- **Library:** Use wmp.dll.

See Also

- [Player Object](#)
- [Settings.getMode](#)
- [Settings.setMode](#)
Player.MouseDown Event

The **MouseDown** event occurs when the user presses a mouse button.

**Syntax**

`MouseDown(nButton, nShiftState, fX, fY)`

**Parameters**

- **`nButton`**
  Number (int) specifying a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

- **`nShiftState`**
  Number (int) specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

- **`fX`**
  Number (long) specifying the x coordinate of the mouse pointer relative to the upper left-hand corner of the control.

- **`fY`**
  Number (long) specifying the y coordinate of the mouse pointer relative to the upper left-hand corner of the control.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.
Player.MouseMove Event

The `MouseMove` event occurs when the mouse pointer is moved.

Syntax

```
MouseMove(nButton, nShiftState, fX, fY)
```

Parameters

- **nButton**
  - Number (int) specifying a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the buttons are pressed.

- **nShiftState**
  - Number (int) specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed.

- **fX**
  - Number (long) specifying the x coordinate of the mouse pointer relative to the upper left-hand corner of the control.

- **fY**
  - Number (long) specifying the y coordinate of the mouse pointer relative to the upper left-hand corner of the control.
Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object

Player.MouseUp Event

The **MouseUp** event occurs when the user releases a mouse button.

Syntax

```
MouseUp(nButton, nShiftState, fX, fY)
```

Parameters

- **nButton**

  Number (int) specifying a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

- **nShiftState**

  Number (int) specifying a bit field with the least significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some,
all, or none of the keys are pressed.

\[ fX \]

**Number (long)** specifying the x coordinate of the mouse pointer relative to the upper left-hand corner of the control.

\[ fY \]

**Number (long)** specifying the y coordinate of the mouse pointer relative to the upper left-hand corner of the control.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Player Object**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.network**

The **network** property retrieves the **Network** object.

**Syntax**

default

```
player.network
```

**Possible Values**
This property is a read-only **Network** object.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Network Object**
- **Player Object**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.newMedia**

The **newMedia** method creates a new **Media** object.

**Syntax**

```
player.newMedia(URL)
```

**Parameters**

- **URL**
  
  *String* containing the URL of the digital media file to create the **Media** object with.

**Return Values**

This method returns a **Media** object.

**Remarks**

The **URL** parameter must not be an empty string or null.

**Requirements**
Player.newPlaylist

The newPlaylist method creates a new Playlist object.

Syntax

```
player.newPlaylist(name, URL)
```

Parameters

- **name**
  
  String containing the name of the new playlist.

- **URL**
  
  String containing the URL of the Windows Media metafile playlist to create the Playlist object with.

Return Values

This method returns a Playlist object.

Remarks

If the URL parameter is set to null or "" (empty string), an empty Playlist object will be created. If the name parameter is set to "", the name in the metafile is used.

The new playlist created with this method is not added to Media Library. To add a new playlist to Media Library, use PlaylistCollection.importPlaylist or PlaylistCollection.newPlaylist. Any leading or trailing
spaces in the playlist name are automatically removed when it is added to Media Library.

Because Media Library allows multiple playlists with the same name, you may want to check for the presence of a playlist with a given name before adding a new one.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- PlaylistCollection.importPlaylist
- PlaylistCollection.newPlaylist
- Windows Media Metafiles

Player.openPlayer

The openPlayer method opens Windows Media Player using the specified URL.

Syntax

player.openPlayer(URL)

Parameters

URL

String representing the URL of the media item to play.

Remarks

This method launches Windows Media Player with the specified URL set as the current media item. If the Player was previously closed in skin mode it will open using the skin last chosen by the user. Otherwise, the Player opens in full mode.
If this method is called from a Player ActiveX control embedded in remote mode, its behavior is identical to the `PlayerApplication.switchToPlayerApplication` method.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- Player Object
- `PlayerApplication.switchToPlayerApplication`

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.OpenPlaylistSwitch Event**

The `OpenPlaylistSwitch` event occurs when a title on a DVD begins playing.

**Syntax**

`OpenPlaylistSwitch(pItem)`

**Parameters**

- `pItem`

`Playlist` object representing the title.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player for Windows XP or later.
Player.openState

The `openState` property retrieves a value indicating the state of the content source.

Syntax

```plaintext
player.openState
```

Possible Values

This property is a read only `Number (long)`.

<table>
<thead>
<tr>
<th>Value</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Undefined</td>
<td>Windows Media Player is in an undefined state.</td>
</tr>
<tr>
<td>1</td>
<td>PlaylistChanging</td>
<td>New playlist is about to be loaded.</td>
</tr>
<tr>
<td>2</td>
<td>PlaylistLocating</td>
<td>Windows Media Player is attempting to locate the playlist. The playlist can be local (<code>Media Library</code> or metafile with a .asx file name extension) or remote.</td>
</tr>
<tr>
<td>3</td>
<td>PlaylistConnecting</td>
<td>Connecting to the playlist.</td>
</tr>
<tr>
<td>4</td>
<td>PlaylistLoading</td>
<td>Playlist has been found and is now being retrieved.</td>
</tr>
<tr>
<td>5</td>
<td>PlaylistOpening</td>
<td>Playlist has been retrieved and is now being parsed and loaded.</td>
</tr>
<tr>
<td>6</td>
<td>PlaylistOpenNoMedia</td>
<td>Playlist is open.</td>
</tr>
<tr>
<td>7</td>
<td>PlaylistChanged</td>
<td>A new playlist has been assigned to <code>currentPlaylist</code>.</td>
</tr>
<tr>
<td>State</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>MediaChanging</td>
<td>New media is about to be loaded.</td>
<td></td>
</tr>
<tr>
<td>MediaLocating</td>
<td>Windows Media Player is locating the media file. The file can be local or remote.</td>
<td></td>
</tr>
<tr>
<td>MediaConnecting</td>
<td>Connecting to the server that holds the media.</td>
<td></td>
</tr>
<tr>
<td>MediaLoading</td>
<td>Media has been located and is now being retrieved.</td>
<td></td>
</tr>
<tr>
<td>MediaOpening</td>
<td>Media has been retrieved and is now being opened.</td>
<td></td>
</tr>
<tr>
<td>MediaOpen</td>
<td>Media is now open.</td>
<td></td>
</tr>
<tr>
<td>BeginCodecAcquisition</td>
<td>Starting codec acquisition.</td>
<td></td>
</tr>
<tr>
<td>EndCodecAcquisition</td>
<td>Codec acquisition is complete.</td>
<td></td>
</tr>
<tr>
<td>BeginLicenseAcquisition</td>
<td>Acquiring a license to play DRM protected content.</td>
<td></td>
</tr>
<tr>
<td>EndLicenseAcquisition</td>
<td>License to play DRM protected content has been acquired.</td>
<td></td>
</tr>
<tr>
<td>BeginIndividualization</td>
<td>Begin DRM Individualization.</td>
<td></td>
</tr>
<tr>
<td>EndIndividualization</td>
<td>DRM individualization has been completed.</td>
<td></td>
</tr>
<tr>
<td>MediaWaiting</td>
<td>Waiting for media.</td>
<td></td>
</tr>
<tr>
<td>OpeningUnknownURL</td>
<td>Opening a URL with an unknown type.</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**

Windows Media Player states are not guaranteed to occur in any particular order. Furthermore, not every state necessarily occurs during a sequence of events. You should not write code that relies upon state order.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player Object](#)
- [Player.OpenStateChange Event](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.
Player.OpenStateChange Event

The OpenStateChange event occurs when the openState property changes value.

Syntax

OpenStateChange(NewState)

Parameters

NewState

Number (long) specifying the new open state. See openState for a table of values.

Remarks

Windows Media Player can go through several open states while it attempts to open a network file, such as locating the server, connecting to the server, and finally opening the file. This event will be fired each time the open state changes.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Windows Media Player states are not guaranteed to occur in any particular order. Furthermore, not every state necessarily occurs during a sequence of events. You should not write code that relies upon state order.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.openState
Player.playerApplication

The `playerApplication` property retrieves the `PlayerApplication` object when a remoted Windows Media Player control is running.

Syntax

```
playerApplication
```

Possible Values

This property is a read-only `PlayerApplication` object or the null value.

Remarks

This property is used only when remoting the Player control. If the value is null, the Player control is not embedded in remote mode.

This property is only accessible in C++ code or in script code in skins through the `playerApplication` global variable.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Global Attributes](#)
- [Player Object](#)
- [PlayerApplication Object](#)
- [Remoting the Windows Media Player Control](#)
Player.PlayerDockedStateChange Event

The `PlayerDockedStateChange` event occurs when a remoted Windows Media Player control docks or undocks.

**Syntax**

```c
PlayerDockedStateChange()
```

**Remarks**

This event occurs only when remoting the Player control.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player Object](#)
- [Remoting the Windows Media Player Control](#)

Player.PlayerReconnect Event

The `PlayerReconnect` event occurs when a remoted Windows Media Player control loses its remote connection to the standalone Player and attempts to reconnect.

**Syntax**
Remarks

This event occurs only when remoting the Player control.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Remoting the Windows Media Player Control

Player.PlaylistChange Event

The PlaylistChange event occurs when a playlist changes.

Syntax

PlaylistChange(Playlist, change)

Parameters

Playlist

Playlist object which changed.

change

Number (long) indicating what type of change occurred to the playlist. Includes one of the following values.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object

© 2000-2003 Microsoft Corporation. All rights reserved.
The **playlistCollection** property retrieves the **PlaylistCollection** object.

**Syntax**

```javascript
player.playlistCollection
```

**Possible Values**

This property is a read-only **PlaylistCollection** object.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player Object](#)
- [PlaylistCollection Object](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.PlaylistCollectionChange Event**

The **PlaylistCollectionChange** event occurs when something changes in the playlist collection.

**Syntax**

```javascript
PlaylistCollectionChange()
```

**Requirements**
Player.PlaylistCollectionPlaylistAdded Event

The `PlaylistCollectionPlaylistAdded` event occurs when a playlist is added to the playlist collection.

**Syntax**

`PlaylistCollectionPlaylistAdded(bstrPlaylistName)`

**Parameters**

`bstrPlaylistName`

String specifying the name of the playlist that was added.

**Remarks**

The name of the playlist that was added can be used to retrieve the corresponding `Playlist` object using the `PlaylistCollection.getByName` method.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.
Library: Use wmp.dll.

See Also

- Player Object
- PlaylistCollection.getByName

Player.PlaylistCollectionPlaylistRemoved Event

The **PlaylistCollectionPlaylistRemoved** event occurs when a playlist is removed from the playlist collection.

**Syntax**

```plaintext
PlaylistCollectionPlaylistRemoved(bstrPlaylistName)
```

**Parameters**

- `bstrPlaylistName`
  
  String specifying the name of the playlist that was removed.

**Remarks**

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

See Also

- Player Object
- PlaylistCollection.getByName
Player.playState

The `playState` property retrieves a value indicating the state of the Windows Media Player operation.

Syntax

`player.playState`

Possible Values

This property is a read-only `Number (long)`.

<table>
<thead>
<tr>
<th>Value</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Undefined</td>
<td>Windows Media Player is in an undefined state.</td>
</tr>
<tr>
<td>1</td>
<td>Stopped</td>
<td>Playback of the current media clip is stopped.</td>
</tr>
<tr>
<td>2</td>
<td>Paused</td>
<td>Playback of the current media clip is paused. When media is paused, resuming playback begins from the same location.</td>
</tr>
<tr>
<td>3</td>
<td>Playing</td>
<td>The current media clip is playing.</td>
</tr>
<tr>
<td>4</td>
<td>ScanForward</td>
<td>The current media clip is fast forwarding.</td>
</tr>
<tr>
<td>5</td>
<td>ScanReverse</td>
<td>The current media clip is fast rewinding.</td>
</tr>
<tr>
<td>6</td>
<td>Buffering</td>
<td>The current media clip is getting additional data from the server.</td>
</tr>
<tr>
<td>7</td>
<td>Waiting</td>
<td>Connection is established, however the server is not sending bits. Waiting for session to begin.</td>
</tr>
<tr>
<td>8</td>
<td>MediaEnded</td>
<td>Media has completed playback and is at its end.</td>
</tr>
<tr>
<td>9</td>
<td>Transitioning</td>
<td>Preparing new media.</td>
</tr>
<tr>
<td>10</td>
<td>Ready</td>
<td>Ready to begin playing.</td>
</tr>
<tr>
<td>11</td>
<td>Reconnecting</td>
<td>Reconnecting to stream.</td>
</tr>
</tbody>
</table>

Remarks

Windows Media Player states are not guaranteed to occur in any particular order. Furthermore, not every state
necessarily occurs during a sequence of events. You should not write code that relies upon state order.

Example Code

The following JScript code shows the use of the `player.playState` property. An HTML text element, named "myText", displays the current status. The player object was created with ID = "Player".

```javascript
// Test whether Windows Media Player is in the playing state.
if (3 == Player.playState)
    myText.value = "Windows Media Player is playing!";
else
    myText.value = "Windows Media Player is NOT playing!";
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.PlayStateChange Event

Player.PlayStateChange Event

The `PlayStateChange` event occurs when there is a change in the `playState` property.

Syntax

`PlayStateChange(NewState)`

Parameters

`NewState`

Number (long) which specifies the new `playState`. See `playState` for a table of possible values.
Remarks

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Windows Media Player states are not guaranteed to occur in any particular order. Furthermore, not every state necessarily occurs during a sequence of events. You should not write code that relies upon state order.

Example Code

The following example demonstrates an event handler for the `Player.playStateChange` event. An HTML text element, named "myText", displays the new play state. The player object was created with ID = "Player".

```javascript
<SCRIPT LANGUAGE = "JScript" FOR = Player EVENT = playStateChange(NewState)>
// Test for the player current state, display a message for each.
switch (NewState){
    case 1:
        myText.value = "Stopped";
        break;
    case 2:
        myText.value = "Paused";
        break;
    case 3:
        myText.value = "Playing";
        break;
    // Other cases go here.
    default:
        myText.value = "";
}
</SCRIPT>
```

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- [Player Object](#)
- [Player.playState](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Player.PositionChange Event

The **PositionChange** event occurs when the current position of the media has been changed.

Syntax

```
PositionChange(oldPosition, newPosition)
```

Parameters

- `oldPosition`
  - **Number (double)** specifying the old position.

- `newPosition`
  - **Number (double)** specifying the new position.

Remarks

This event is not raised routinely during playback. It only occurs when something actively changes the current position of the playing media, such as the user moving the seek handle or code specifying a value for `Controls.currentPosition`.

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

Requirements

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

See Also

- [Player Object](#)
Player.ScriptCommand Event

The **ScriptCommand** event occurs when a synchronized command or URL is received.

Syntax

```
ScriptCommand(scType, Param)
```

**Parameters**

- **scType**
  - String specifying the type of script command.
- **Param**
  - String specifying the script command.

**Remarks**

Commands can be embedded among the sounds and images of a Windows Media file. The commands are a pair of Unicode strings associated with a designated time in the stream. When the stream reaches the time associated with the command, the Windows Media Player control sends a **ScriptCommand** event with two parameters. One parameter specifies the type of command being sent, and the other parameter specifies the command. The type of parameter is used to determine how the command parameter is processed. Any type of command can be embedded in a Windows Media stream to be handled by the **ScriptCommand** event.

One type of command is a **URL** (Uniform Resource Locator). **URL**-type commands received by the Player control are invoked automatically in your default Internet browser if the **Settings.invokeURLs** property is set to true. You can use the **Settings.defaultFrame** property to specify the target frame in which the Web page appears.

The URL sent to Windows Media Player is processed relative to the base URL specified by the **Settings.baseURL** property. The base URL is concatenated with the relatively specified URL, resulting in a fully specified URL that is passed as the command parameter by the **ScriptCommand** event.

The Player control always processes incoming URL-type commands in the following manner:

1. A URL-type command is received.
2. **Settings.baseURL** is used to create a full URL from the relative URL specified in the script command.
3. **ScriptCommand** is called.
4. After **ScriptCommand** returns, **Settings.invokeURLs** is checked.
5. If **Settings.invokeURLs** is true and the command is a URL-type, the specified URL is invoked. If **Settings.invokeURLs** is false or if the command is not a URL-type, the command is ignored.
When authoring a Windows Media file, you can specify which frame the new URL is displayed in by concatenating two ampersands and the name of the frame in the parameter field. The example below illustrates typical ScriptCommand parameters. It specifies that the URL mypage must be launched in the myframe frame.

```plaintext
scType = "URL"
Param = http://myweb/mypage.html&&myframe
```

The following table lists script command types that are automatically processed by Windows Media Player.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPTION</td>
<td>The control displays the associated text in the DIV specified by ClosedCaption.captioningID.</td>
</tr>
<tr>
<td>EVENT</td>
<td>Tells the control to execute instructions defined for the specified event.</td>
</tr>
<tr>
<td>FILENAME</td>
<td>The control resets its URL property, attempts to open the specified file, and begins playing the new stream immediately.</td>
</tr>
<tr>
<td>OPENEVENT</td>
<td>Buffers the associated EVENT type command for timely execution of the EVENT script.</td>
</tr>
<tr>
<td>SYNCHRONIZEDLYRICLYRIC</td>
<td>The Param parameter contains the synchronized lyric text. Windows Media Player displays the lyric text in the closed caption area of the Now Playing feature.</td>
</tr>
<tr>
<td>TEXT</td>
<td>The control displays the associated text in the DIV specified by ClosedCaption.captioningID.</td>
</tr>
<tr>
<td>URL</td>
<td>The control automatically opens the URL specified using the default Internet browser if the Settings.invokeURLs property is set to true.</td>
</tr>
</tbody>
</table>

You can embed any other type of command as long as you provide reciprocal code to handle the command. Though unknown commands are ignored by the Player control, they are still handed off to the ScriptCommand event.

The ScriptCommand event is not called if the file is being scanned (fast-forwarded or fast-reversed).

The value of event parameters is specified by the player, and can be accessed or passed to a method in an imported JScript file using the parameter name given. This parameter name must be typed exactly as shown, including capitalization.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**
Player.settings

The settings property retrieves the Settings object.

Syntax

player.settings

Possible Values

This property is a read-only Settings object.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

• Player Object
• Settings Object
Player.status

The status property retrieves a value indicating the status of Windows Media Player.

Syntax

player.status

Possible Values

This property is a read-only String.

Remarks

The values contained in this property are subject to change at any time, and should be used for display purposes only.

The StatusChange event is fired whenever this property changes value.

Requirements

Version: Windows Media Player 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.StatusChange Event
Player.StatusChange Event

The StatusChange event occurs when the status property changes value.

Syntax

StatusChange()

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object
- Player.status

Player/stretchToFit

The stretchToFit property specifies or retrieves a value indicating whether video displayed by the Windows Media Player control automatically sizes to fit the video window, when the video window is larger than the dimensions of the video image.

Syntax

player/stretchToFit

Possible Values

This property is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The video will stretch to fit the window.</td>
</tr>
</tbody>
</table>
Remarks

When `stretchToFit` is set to true, the Windows Media Player control maintains the original aspect ratio of the video. If the aspect ratio of the video does not match the aspect ratio of the video window, black mask areas may appear on either the top and the bottom, or left and right, of the video image.

This property applies to the Windows Media Player control only when embedded in a Web page.

Requirements

Version: Windows Media Player version 7.1 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player Object

**Player.SwitchedToControl Event**

The `SwitchedToControl` event occurs when a remoted Windows Media Player control switches to the docked state.

Syntax

```
SwitchedToControl()
```

Remarks

This event occurs only when remoting the Player control, and only occurs for the Player control instance being switched to.

Requirements

Version: Windows Media Player 9 Series or later.
The **SwitchedToPlayerApplication** event occurs when a remoted Windows Media Player control switches to the full mode of the Player.

### Syntax

```
SwitchedToPlayerApplication()
```

### Remarks

This event occurs only when remoting the Player control.

### Requirements

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.
Player.uiMode

The `uiMode` property specifies or retrieves a value indicating which controls are shown in the user interface.

**Syntax**

`player.uiMode`

**Possible Values**

This property is a read/write String.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Audio example</th>
<th>Video example</th>
</tr>
</thead>
<tbody>
<tr>
<td>invisible</td>
<td>Player is embedded without any visible user interface (controls, video or visualization window).</td>
<td>(Nothing is displayed)</td>
<td>(Nothing is displayed)</td>
</tr>
<tr>
<td>none</td>
<td>Player is embedded without controls, and with only the video or visualization window displayed.</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>mini</td>
<td>Player is embedded with the status window, play/pause, stop, mute, and volume controls shown in addition to the video or visualization window.</td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>full</td>
<td>Default. Player is embedded with the status window, seek bar, play/pause, stop, mute, next, previous, fast forward, fast reverse, and volume controls in addition to the video or visualization window.</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>custom</td>
<td>Player is embedded with a custom user interface. Can only be used in C++ programs.</td>
<td>(Custom user interface is displayed.)</td>
<td>(Custom user interface is displayed.)</td>
</tr>
</tbody>
</table>

**Remarks**

This property specifies the appearance of the embedded Windows Media Player. When `uiMode` is set to "none", "mini", or "full", a window is present for the display of video clips and audio visualizations. This window can
be hidden in mini or full mode by setting the **height** attribute of the **OBJECT** tag to 40, which is measured from the bottom, and leaves the controls portion of the user interface visible. If no embedded interface is desired, set both the **width** and **height** attributes to zero.

If **uiMode** is set to "invisible", no user interface is displayed, but space is still reserved on the page as specified by **width** and **height**. This is useful for retaining page layout when **uiMode** can change. Additionally, the reserved space is transparent, so any elements layered behind the control will be visible.

If **uiMode** is set to "full" or "mini", the Player displays transport controls in full-screen mode. If **uiMode** is set to "none", no controls are displayed in full-screen mode.

If the window is visible and audio content is being played, the visualization displayed will be the one most recently used in Windows Media Player.

If **uiMode** is set to "custom" in a C++ program that implements **IWMPRemoteMediaServices**, the skin file indicated by **IWMPRemoteMediaServices::GetCustomUIMode** is displayed.

During full-screen playback, Windows Media Player hides the mouse cursor when **enableContextMenu** equals false and **uiMode** equals "none".

### Example Code

The following example creates an HTML SELECT element that allows the user to change the user interface for an embedded Player object. The Player object was created with ID = "Player".

```html
<!-- Create an HTML SELECT element. -->
<SELECT ID = UI LANGUAGE="JScript"
    /* Specify the UI mode the user selects. */
    onChange = "Player.uiMode = UI.value">
    /* These are the four UI mode options. */
    <OPTION VALUE="invisible">Invisible
    <OPTION VALUE="none">No Controls
    <OPTION VALUE="mini">Mini Player
    <OPTION VALUE="full">Full Player
</SELECT>
```

### Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

### See Also

- **IWMPRemoteMediaServices** Interface
- **IWMPRemoteMediaServices::GetCustomUIMode**
- **Player Object**
Player.URL

The URL property specifies or retrieves the name of the clip to play.

Syntax

player.URL

Possible Values

This property is a read/write String with no default value.

Remarks

This property can only be set to a URL in a security zone that is the same or is less restrictive than the security zone of the calling program or Web page.

Applications that open media from behind a firewall will have better performance when opening media if the address is specified using the domain name server (DNS) name instead of the IP address.

Do not call this method from event handler code. Calling Player.URL from an event handler may yield unexpected results.

Example Code

The following example creates an HTML TEXT input element and an HTML BUTTON input element. The TEXT element allows the user to type a path to specify a digital media file to play. The BUTTON element executes JScript that opens the file and starts the Player. The Player object was created with ID = "Player".

<!-- Create an INPUT control to get a file path from the user. -->
<INPUT Type = "TEXT" ID = "inputURL">

<!-- Create a BUTTON control to execute the script. -->
<INPUT Type = "BUTTON" ID = "openMedia" VALUE = "Open Media"
        onClick = "
            /* Specify the URL obtained from user input. */
            Player.URL = inputURL.value;

            /* Start the Player. */
            Player.controls.play();
        "">

Requirements
**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player Object](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Player.versionInfo**

The `versionInfo` property retrieves a `String` value specifying the version of the Windows Media Player.

**Syntax**

```javascript
player.versionInfo
```

**Possible Values**

This property is a read-only `String` in the following format: "X.0.0.YYYY" where `X` represents the major version number and `YYYY` represents the build number.

**Example Code**

The following example creates an HTML BUTTON element that, when clicked, displays a message box containing the version information for the Windows Media Player object. The Player object was created with ID = "Player".

```html
<INPUT TYPE = "BUTTON"  ID = "VERSION"  VALUE = "Show Version"
onclick = "
/* Build the message containing the version info. */
var message = 'Windows Media Player Version: ';
message += '\n';
message += Player.versionInfo;

/* Display the message box. */
alert(message);
"
">
```

**Requirements**
**Player.windowlessVideo**

The `windowlessVideo` property specifies or retrieves a value indicating whether the Windows Media Player control renders video in windowless mode.

**Syntax**

```javascript
player.windowlessVideo
```

**Possible Values**

This property is a **Boolean** that is specified at design time and is read-only thereafter.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The video is rendered in windowless mode.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The video is rendered in windowed mode.</td>
</tr>
</tbody>
</table>

**Remarks**

By default, an embedded Player object renders video in its own window within the client area. When `windowlessVideo` is set to true, the Player object renders video directly in the client area, so you can apply special effects or layer the video with text.

The `windowlessVideo` property is not supported for Netscape Navigator. Setting a value for this property in Navigator may yield unexpected results.

**Requirements**
PlayerApplication Object

The PlayerApplication object provides a way to coordinate switching between a remoted Player control and the full mode of the Player. This object is used only with C++ programs that implement the IWMPRemoteMediaServices interface and embed the Player control in remote mode. Skin files used as custom interfaces for the remoted control access this object in script code through the playerApplication global attribute.

The PlayerApplication object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasDisplay</td>
<td>Retrieves a value indicating whether video can display through the remoted Player control.</td>
</tr>
<tr>
<td>playerDocked</td>
<td>Retrieves a value indicating whether the Player is in a docked state.</td>
</tr>
</tbody>
</table>

The PlayerApplication object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switchToControl</td>
<td>Switches a remoted Player control to the docked state.</td>
</tr>
<tr>
<td>switchToPlayerApplication</td>
<td>Switches a remoted Player control to the full mode of the Player.</td>
</tr>
</tbody>
</table>

The PlayerApplication object is accessed through the following property.

| Object | Property |
The `hasDisplay` property retrieves a value indicating whether video can display through the remoted Player control.

**Syntax**

```plaintext
playerApplication.hasDisplay
```

**Possible Values**

This property is a read-only `Boolean`.

**Remarks**

This property is used only when remoting the Player control.

Several Player controls can be running remotely at the same time, but video can only display in one location at a time, either in the full mode of the Player or in one of the remoted controls. Use this property to determine whether the current control is the one through which video can be displayed.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- `PlayerApplication Object`
The `playerDocked` property retrieves a value indicating whether the Player is in a docked state.

**Syntax**

```
playerApplication.playerDocked
```

**Possible Values**

This property is a read-only `Boolean`.

**Remarks**

This property is used only when remoting the Player control.

**Requirements**

- **Version:** Windows Media Player 9 Series or later.
- **Header:** Defined in `wmp.idl`; include `wmp.h`.
- **Library:** Use `wmp.dll`.

See Also

- `PlayerApplication` Object
- Remoting the Windows Media Player Control
PlayerApplication.switchToControl

The `switchToControl` method switches a remoted Player control to the docked state.

Syntax

```javascript
playerApplication.switchToControl()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

This method is used only when remoting the Player control.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- `PlayerApplication Object`
- `Remoting the Windows Media Player Control`

© 2000-2003 Microsoft Corporation. All rights reserved.
PlayerApplication.switchToPlayerApplication

The **switchToPlayerApplication** method switches a remoted Player control to the full mode of the Player.

**Syntax**

```csharp
PlayerApplication.switchToPlayerApplication()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

This method is used only when remoting the Player control.

**Requirements**

- **Version**: Windows Media Player 9 Series or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- [PlayerApplication Object](#)
- [Remoting the Windows Media Player Control](#)

---

**Playlist Object**

The **Playlist** object provides a way to organize media items in a list for easy manipulation using the following properties and methods.
The **Playlist** object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeCount</td>
<td>Retrieves the number of attributes associated with the playlist.</td>
</tr>
<tr>
<td>attributeName</td>
<td>Retrieves the name of an attribute specified by an index.</td>
</tr>
<tr>
<td>count</td>
<td>Retrieves the number of items in the playlist.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies or retrieves the name of the playlist.</td>
</tr>
</tbody>
</table>

The **Playlist** object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendItem</td>
<td>Adds a media item to the end of the playlist.</td>
</tr>
<tr>
<td>clear</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>getItemInfo</td>
<td>Retrieves the value of a playlist attribute.</td>
</tr>
<tr>
<td>insertItem</td>
<td>Inserts a media item into the playlist at the specified location.</td>
</tr>
<tr>
<td>isIdentical</td>
<td>Retrieves a value indicating whether the supplied <strong>Playlist</strong> object is</td>
</tr>
<tr>
<td></td>
<td>identical to the current one.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves the media item at the specified index.</td>
</tr>
<tr>
<td>moveItem</td>
<td>Changes the location of an item in the playlist.</td>
</tr>
<tr>
<td>removeItem</td>
<td>Removes the specified item from the playlist.</td>
</tr>
<tr>
<td>setItemInfo</td>
<td>Specifies the value of a playlist attribute.</td>
</tr>
</tbody>
</table>

The **Playlist** object is accessed through the following properties and methods.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property or method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cdrom</td>
<td>playlist</td>
</tr>
<tr>
<td>MediaCollection</td>
<td>getAll, getByAlbum, getByAttribute, getByAuthor, getByGenre, getByName</td>
</tr>
<tr>
<td>Player</td>
<td>currentPlaylist, newPlaylist</td>
</tr>
<tr>
<td>PlaylistArray</td>
<td>item</td>
</tr>
<tr>
<td>PlaylistCollection</td>
<td>newPlaylist</td>
</tr>
</tbody>
</table>

Because it is the most common means of access, **player.currentPlaylist** is used for purposes of illustration in the reference syntax sections.

**See Also**

- **Object Model Reference**
**Playlist.appendItem**

The `appendItem` method adds a media item to the end of the playlist.

**Syntax**

```javascript
player.currentPlaylist.appendItem(item)
```

**Parameters**

`item`

*Media* object to be added.

**Return Values**

This method does not return a value.

**Remarks**

To use this method, full access to *Media Library* is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example uses `Playlist.appendItem` to add the current media item to the playlist named "ThreeList". The player object was created with ID="Player".

```javascript
// Get the current media item.
var currMedia = Player.currentMedia;

// Get the playlist object by name.
var plThreeList = Player.playlistCollection.getByName("ThreeList").item(0);

// Append the media item to the playlist.
plThreeList.appendItem(currMedia);
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.
Playlist.attributeCount

The `attributeCount` property retrieves the number of attributes associated with the playlist.

Syntax

```javascript
player.currentPlaylist.attributeCount
```

Possible Values

This property is a read-only `Number (long)`.

Remarks

Because playlists can come from many different sources, they can have several different sets of properties. This method retrieves the total number of properties available so that the other methods of the `Playlist` object can access them.

To retrieve the value of this property, read access to `Media Library` is required. For more information, see `Media Library Access`.

For information about the attributes supported by Windows Media Player, see `Available Attributes`.

Example

The following JScript example illustrates how various properties and methods of the `Playlist` and `Media` objects are used.

```javascript
function onLoad() {
    var display;
    var pl = player.currentPlaylist;
```
pl.setItemInfo("custom playlist attribute", "changed");
pl.item(0).setItemInfo("new custom attribute", "5");

display = pl.attributeCount + " Playlist Attributes:\r\n";

for (var i = 0; i < pl.attributeCount; ++i) {
    display = display + pl.attributeName(i) +":\r"
    display = display + pl.getItemInfo(pl.attributeName(i)) +\r"
}

for (var j = 0; j < pl.count; ++j) {
    display = display +\rTrack " + j +\r"
    display = display + pl.item(j).attributeCount + " Attributes:\r\n"

    for (var k = 0; k < pl.item(j).attributeCount; ++k) {
        var it = pl.item(j); // Media object
        display = display + it.getAttributeName(k) +":\r"
        display = display + it.getItemInfo(it.getAttributeName(k)) +\r"
    }
}

myText.value = display;

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Playlist Object
- Playlist.attributeName
- Playlist.getItemInfo
- Playlist.setItemInfo
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.
The **attributeName** property retrieves the name of an attribute specified by an index.

**Syntax**

```
player.currentPlaylist.attributeName(index)
```

**Parameters**

- **index**
  
  Number (**long**) containing the index.

**Possible Values**

This property is a read-only **String**.

**Remarks**

The number of attributes is retrieved with the **attributeCount** property. Given an index, **attributeName** returns a **String** that can be used in conjunction with **setItemInfo** and **getItemInfo** to specify and retrieve the value for an attribute.

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

**Example**

See the **attributeCount** property for a sample illustrating the use of this property.

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [Playlist Object](#)
- [Playlist.attributeCount](#)
- [Playlist.getItemInfo](#)
- [Playlist.setItemInfo](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
Playlist.count

The **count** property retrieves the number of media items in the playlist.

**Syntax**

```
player.currentPlaylist.count
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. For more information, see **Media Library Access**.

**Example**

See the **attributeCount** property for an example illustrating the use of this property.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- **Playlist Object**
- **Settings.mediaAccessRights**
- **Settings.requestMediaAccessRights**
Playlist.getItemInfo

The `getItemInfo` method retrieves the value of a playlist attribute.

Syntax

```
player.currentPlaylist.getItemInfo(name)
```

Parameters

- `name`

String containing the name of the attribute to be retrieved. For information about the attributes supported by Windows Media Player, see Available Attributes.

Return Values

This method returns a String.

Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Example

See the `attributeCount` property for a sample illustrating the use of this property.

Requirements

- **Version:** Windows Media Player version 7.0 or later.
- **Header:** Defined in wmp.idl; include wmp.h.
- **Library:** Use wmp.dll.

See Also

- Playlist Object
- Playlist.setItemInfo
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
Playlist.insertItem

The `insertItem` method inserts a media item into the playlist at the specified location.

Syntax

```
player.currentPlaylist.insertItem(index, item)
```

Parameters

- **index**
  
  *Number (long)* containing the index at which to add the item.

- **item**
  
  *Media* object to insert.

Return Values

This method does not return a value.

Remarks

All items after the inserted item will have their index numbers increased by one.

To use this method, full access to *Media Library* is required. For more information, see [Media Library Access](#).

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Playlist Object](#)
- [Playlist.item](#)
- [Playlist.removeItem](#)
The `isIdentical` method retrieves a value indicating whether the supplied `Playlist` object is identical to the current one.

**Syntax**

```plaintext
player.currentPlaylist.isIdentical(playlist)
```

**Parameters**

- `playlist`

  `Playlist` object to compare to the current one.

**Return Values**

This method returns a `Boolean`.

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- [Playlist Object](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.
Playlist.item

The **item** method retrieves the media item at the specified index.

**Syntax**

```
player.currentPlaylist.item(index)
```

**Parameters**

`index`

**Number (long)** containing the index.

**Return Values**

This method returns a **Media** object.

**Remarks**

To use this method, read access to **Media Library** is required. For more information, see Media Library Access.

**Example Code**

The following JScript example uses **Playlist.item** to retrieve a particular media item from the current playlist based on a user selection. An HTML select element was created with name = "weblist", and filled with the titles from the current playlist. The player object was created with ID = "Player".

```
// Store the value of the selected item in the list box
// using the selectedIndex property.
var index = weblist.selectedIndex;

// Store the corresponding media object from the current playlist.
var listItem = Player.currentPlaylist.item(index);

// Set the player URL using the listItem variable.
Player.URL = listItem.sourceURL;
```

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.
**PlayList.moveItem**

The `moveItem` method changes the location of an item in the playlist.

**Syntax**

```javascript
player.currentPlaylist.moveItem(oldIndex, newIndex)
```

**Parameters**

- `oldIndex`
  
  Number (`long`) containing the old index.

- `newIndex`
  
  Number (`long`) containing the new index.

**Return Values**

This method does not return a value.

**Remarks**

Playlists stored in **Media Library** can change outside your control. Be sure to monitor and handle all appropriate playlist-related events so that the order of items in the playlist does not change unexpectedly.

To use this method, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**
Playlist.name

The **name** property specifies or retrieves the name of the playlist.

**Syntax**

```cpp
player.currentPlaylist.name
```

**Possible Values**

This property is a read/write **String**.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. To specify the value, full access is required. For more information, see **Media Library Access**.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**
The `removeItem` method removes the specified item from the playlist.

**Syntax**

```
player.currentPlaylist.removeItem(item)
```

**Parameters**

- `item`  
  Media object to be removed.

**Return Values**

This method does not return a value.

**Remarks**

If the item removed is the currently playing track (`Player.currentMedia`), playback stops and the next item in the playlist becomes the current one. If there is no next item, the previous item is used, or if there are no other items, then `Player.currentMedia` is set to `NULL`.

To use this method, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**

- **Version:** Windows Media Player version 7.0 or later.

- **Header:** Defined in wmp.idl; include wmp.h.

- **Library:** Use wmp.dll.

See Also
Playlist.setItemInfo

The `setItemInfo` method specifies the value of a playlist attribute.

Syntax

```javascript
player.currentPlaylist.setItemInfo(name, value)
```

Parameters

- **name**
  
  **String** containing the name of the attribute to be set. For information about the attributes supported by Windows Media Player, see [Available Attributes](#).

- **value**
  
  **String** containing the new value for the attribute.

Return Values

This method does not return a value.

Remarks

A special use of the `setItemInfo` method is to sort the items in the playlist, using the `SortAttribute` attribute. The following JScript example sorts a playlist by the values of the `UserLastPlayedTime` attribute. The variable `playlist` is a reference to a `Playlist` object.

```javascript
playlist.setItemInfo("SortAttribute", "UserLastPlayedTime")
```

To use this method, full access to [Media Library](#) is required. For more information, see [Media Library Access](#).
Example

See the attributeCount property for a sample illustrating the use of this property.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Playlist Object
- Playlist.getItemInfo
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

PlaylistArray Object

The PlaylistArray object provides a way to access playlists by index number.

The PlaylistArray object supports the following property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Retrieves the number of playlists in the playlist array.</td>
</tr>
</tbody>
</table>

The PlaylistArray object supports the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Retrieves the Playlist object at the given index.</td>
</tr>
</tbody>
</table>

The PlaylistArray object is accessed through the following methods.

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
</tr>
</thead>
</table>

© 2000-2003 Microsoft Corporation. All rights reserved.
For purposes of illustration, `player.playlistCollection.getAll()` is used in the reference syntax sections.

See Also

- **Object Model Reference**

### PlaylistArray.count

The **count** property retrieves the number of playlists in the playlist array.

**Syntax**

`player.playlistCollection.getAll().count`

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

To retrieve the value of this property, read access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

See Also

- **PlaylistArray Object**
- **Settings.mediaAccessRights**
- **Settings.requestMediaAccessRights**
PlaylistArray.item

The item method retrieves the playlist at the given index.

Syntax

player.playlistCollection.getAll().item(index)

Parameters

index

Number (long) containing the index of the playlist to be retrieved.

Return Values

This method returns a Playlist object.

Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- PlaylistArray Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
PlaylistCollection Object

The PlaylistCollection object provides a way to organize your playlists.

The PlaylistCollection object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAll</td>
<td>Retrieves a PlaylistArray object containing all the playlists in Media Library.</td>
</tr>
<tr>
<td>getName</td>
<td>Retrieves a PlaylistArray object containing playlists with the specified name, if any exist.</td>
</tr>
<tr>
<td>importPlaylist</td>
<td>Adds a static playlist to Media Library.</td>
</tr>
<tr>
<td>isDeleted</td>
<td>Retrieves a value indicating whether the specified playlist is in the deleted items folder.</td>
</tr>
<tr>
<td>newPlaylist</td>
<td>Creates a new playlist in Media Library.</td>
</tr>
<tr>
<td>remove</td>
<td>Removes a playlist from Media Library.</td>
</tr>
<tr>
<td>setDeleted</td>
<td>Moves a playlist to the deleted items folder.</td>
</tr>
</tbody>
</table>

The PlaylistCollection object is accessed through the following property.

Object Property
Player playlistCollection

See Also
- Object Model Reference
PlaylistCollection.getAll

The `getAll` method retrieves a `PlaylistArray` object containing all the playlists in Media Library.

Syntax

```javascript
player.playlistCollection.getAll()
```

Parameters

This method takes no parameters.

Return Values

This method returns a `PlaylistArray` object.

Remarks

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- `PlaylistArray Object`
- `PlaylistCollection Object`
- `Settings.mediaAccessRights`
- `Settings.requestMediaAccessRights`
PlaylistCollection.getByName

The **getByName** method retrieves a **PlaylistArray** object containing playlists with the specified name, if any exist.

**Syntax**

`player.playlistCollection.getByName(name)`

**Parameters**

- `name` String containing the name of the playlists to be retrieved.

**Return Values**

This method returns a **PlaylistArray** object.

**Remarks**

Use **PlaylistArray.count** to determine whether a playlist exists. If **count** is zero, a playlist does not exist.

To use this method, read access to **Media Library** is required. For more information, see Media Library Access.

**Example Code**

The following JScript example uses `playlistCollection.getByName` to check the `playlistCollection` object for a playlist named "ThreeList". If the "Threelist" playlist exists, `getByName` sets "ThreeList" as the current playlist. The player object was created with the ID = "Player".

```javascript
//Retrieve the count of the playlists named "ThreeList".
var Checkit = Player.playlistCollection.getByName("ThreeList").count;

//Since duplicate playlist names are allowed, the count returned
//will be either zero (no playlist) or greater than zero
//(playlist exists).
if (Checkit > 0){
    //Retrieve a playlistArray object containing "ThreeList". Assume that
    //there is only one playlist with that name, and assign it to the
    //current playlist.
    Player.currentPlaylist = Player.playlistCollection.getByName("ThreeList").item(0);
}
```

**Requirements**
Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- PlaylistArray Object
- PlaylistArray.count
- PlaylistCollection Object
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

**PlaylistCollection.importPlaylist**

The **importPlaylist** method adds a static playlist to **Media Library**.

Syntax

```plaintext
player.playlistCollection.importPlaylist(playlist)
```

Parameters

`playlist`

**Playlist** object to be added.

Return Values

This method returns the **Playlist** object that was added.

Remarks

Playlists that do not contain any media items cannot be added to **Media Library** by using this method. To create an empty playlist in **Media Library**, use the **newPlaylist** method. You can then fill the resulting playlist with media items by using **Playlist.appendItem** or **Playlist.insertItem**.

If you pass this method an auto playlist, the query is executed once and the result is added to **Media Library** as
a static playlist. To add an auto playlist to Media Library and preserve its automatic behavior, use MediaCollection.add. For more information, see Static and Auto Playlists.

To use this method, full access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Managing Playlists
- MediaCollection Object
- MediaCollection.add
- Playlist.appendItem
- Playlist.insertItem
- PlaylistCollection Object
- PlaylistCollection.newPlaylist
- PlaylistCollection.remove
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**PlaylistCollection.isDeleted**

The isDeleted method retrieves a value indicating whether the specified playlist is in the deleted items folder.

**Syntax**

`player.playlistCollection.isDeleted(playlist)`

**Parameters**

`playlist`

The Playlist object to be searched for.
Return Values

This method returns a **Boolean**.

Requirements

**Version:** Windows Media Player version 7.0, Windows Media Player version 7.1, or Windows Media Player for Windows XP.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

This method is not supported for Windows Media Player 9 Series or later.

See Also

- **PlaylistCollection Object**

PlaylistCollection.newPlaylist

The **newPlaylist** method creates a new playlist in **Media Library**.

Syntax

```plaintext
player.playlistCollection.newPlaylist(name)
```

Parameters

- **name**
  
  String containing the name of the playlist to be created.

Return Values

This method returns a **Playlist** object.

Remarks
This method creates an empty playlist in **Media Library**. To fill the playlist with media items, use `Playlist.appendItem` or `Playlist.insertItem`.

Multiple playlists having the same name are permitted in **Media Library**. To avoid creating a duplicate playlist name with this method, use `getByName` and `PlaylistArray.count` to determine whether a playlist with a particular name already exists.

Leading and trailing spaces are not permitted in playlist names, and are automatically removed from the value specified for the `name` parameter.

To use this method, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Example Code**

The following JScript example creates a new empty playlist called "ThreeList". The player object was created with ID="Player".

```javascript
//Add a new empty playlist, named ThreeList, to the playlist collection.
var NewList = Player.playlistCollection.newPlaylist("ThreeList");
```

**Requirements**

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- [MediaCollection.add](#)
- [Playlist.appendItem](#)
- [Playlist.insertItem](#)
- [PlaylistArray.count](#)
- [PlaylistCollection Object](#)
- [PlaylistCollection.getByName](#)
- [PlaylistCollection.importPlaylist](#)
- [PlaylistCollection.remove](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)
PlaylistCollection.remove

The `remove` method removes a playlist from **Media Library**.

**Syntax**

```c
player.playlistCollection.remove(playlist)
```

**Parameters**

- `playlist`

The **Playlist** object to be removed.

**Return Values**

This method does not return a value.

**Remarks**

To use this method, full access to **Media Library** is required. For more information, see [Media Library Access](#).

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in `wmp.idl`; include `wmp.h`.
- **Library**: Use `wmp.dll`.

**See Also**

- [PlaylistCollection Object](#)
- [PlaylistCollection.importPlaylist](#)
- [PlaylistCollection.newPlaylist](#)
- [Settings.mediaAccessRights](#)
- [Settings.requestMediaAccessRights](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
**PlaylistCollection.setDeleted**

The **setDeleted** method moves a playlist to the deleted items folder.

**Syntax**

```c
player.playlistCollection.setDeleted(playlist, true)
```

**Parameters**

- **playlist**
  The **Playlist** object to be moved.

- **true**
  Always specify this value.

**Return Values**

This method does not return a value.

**Requirements**

**Version:** Windows Media Player version 7.0, Windows Media Player version 7.1, or Windows Media Player for Windows XP. This method is not supported for Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [PlaylistCollection Object](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Settings Object**

The **Settings** object provides a way to modify various Media Player settings using the following properties and
The **Settings** object supports the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>autoStart</code></td>
<td>Specifies or retrieves a value indicating whether the current media item begins playing automatically.</td>
</tr>
<tr>
<td><code>balance</code></td>
<td>Specifies or retrieves the current stereo balance.</td>
</tr>
<tr>
<td><code>baseURL</code></td>
<td>Specifies or retrieves the base URL used for relative path resolution with URL-type script commands that are embedded in media files.</td>
</tr>
<tr>
<td><code>defaultAudioLanguage</code></td>
<td>Retrieves the locale identifier (LCID) of the default audio language specified in Windows Media Player.</td>
</tr>
<tr>
<td><code>defaultFrame</code></td>
<td>Specifies or retrieves the name of the frame used to display a URL that is received in a <strong>ScriptCommand</strong> event.</td>
</tr>
<tr>
<td><code>enableErrorDialogs</code></td>
<td>Specifies or retrieves a value indicating whether error dialog boxes are shown automatically.</td>
</tr>
<tr>
<td><code>invokeURLs</code></td>
<td>Specifies or retrieves a value indicating whether URL events should launch a Web browser.</td>
</tr>
<tr>
<td><code>isAvailable</code></td>
<td>Retrieves whether a specified type of information is available or a specified action can be performed.</td>
</tr>
<tr>
<td><code>mediaAccessRights</code></td>
<td>Retrieves a value indicating the rights currently granted for <strong>Media Library</strong> access.</td>
</tr>
<tr>
<td><code>mute</code></td>
<td>Specifies or retrieves a value indicating whether audio is muted.</td>
</tr>
<tr>
<td><code>playCount</code></td>
<td>Specifies or retrieves the number of times a media item will play.</td>
</tr>
<tr>
<td><code>rate</code></td>
<td>Specifies or retrieves the current playback rate.</td>
</tr>
<tr>
<td><code>volume</code></td>
<td>Specifies or retrieves the current volume.</td>
</tr>
</tbody>
</table>

The **Settings** object supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getMode</code></td>
<td>Determines whether the loop mode or shuffle mode is active.</td>
</tr>
<tr>
<td><code>requestMediaAccessRights</code></td>
<td>Requests a specified level of access to <strong>Media Library</strong>.</td>
</tr>
<tr>
<td><code>setMode</code></td>
<td>Sets the loop mode or shuffle mode to active or inactive.</td>
</tr>
</tbody>
</table>

The **Settings** object is accessed through the following property.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td>settings</td>
</tr>
</tbody>
</table>
The `autoStart` property specifies or retrieves a value indicating whether the current media item begins playing automatically.

### Syntax

```javascript
player.settings.autoStart
```

### Possible Values

This property is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default (see Remarks). The media item begins playing automatically.</td>
</tr>
<tr>
<td>false</td>
<td>The media item does not begin playing automatically.</td>
</tr>
</tbody>
</table>

### Remarks

If `autoStart` is set to true, the media clip will begin playing as soon as `Player.URL`, `Player.currentPlaylist`, or `Player.currentMedia` is set. Otherwise, it will not start playing until the `Controls.play` method is called.

Because the `autoStart` property for the full mode of the Player can be set globally by external events (such as loading a CD), there is no reliable default value for skins and remoted Player controls. This is because the playback engine of the full Player is used in both cases. You should set `autoStart` to false immediately before you set `Player.URL`, `Player.currentPlaylist`, or `Player.currentMedia` in skins and remoted Player controls if you wish to ensure that the media item does not start playing immediately. Also, unless you set `autoStart` to true immediately before specifying a media item, you should not rely on this setting as a substitute for using the `Controls.play` method.

### Example Code

The following example creates an HTML CHECKBOX element that allows the user to specify whether the Player control plays the current media item automatically. The Player object was created with ID = "Player".

---

**See Also**

- [Object Model Reference](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player.currentMedia
- Player.URL
- Settings Object

Settings.balance

The balance property specifies or retrieves the current stereo balance.

Syntax

player.settings.balance

Possible Values

This property is a read/write Number (long) ranging from –100 to 100 with a default value of zero.

Remarks

The default value zero indicates that the sound is balanced equally between the left and right speakers. A value of –100 indicates that all sound is going to the left speaker, while a value of 100 indicates that all sound is going to the right speaker.

Requirements
Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Settings Object

Settings.baseURL

The baseURL property specifies or retrieves the base URL used for relative path resolution with URL-type script commands that are embedded in media files.

Syntax

player.settings.baseURL

Possible Values

This property is a read/write String.

Remarks

This property specifies the base HTTP URL that is passed as the command parameter by the ScriptCommand event. The base URL is concatenated with the relative URL as follows:

1. A trailing forward slash (/) is added to the value of the baseURL property.
2. A leading period, backward slash, or forward slash (., \, and /) are deleted from the relative URL.
3. The relative URL is added to the end of the base URL.
4. All slashes in the resulting fully qualified URL are pointed in the same direction (converted to forward or backward slashes) based on the direction of the first slash character encountered in the new URL.

Note: The player control does not support the use of two periods (..) in the relative URL to indicate the parent of the current location.

Requirements
**Settings.defaultAudioLanguage**

The **defaultAudioLanguage** property retrieves the LCID of the default audio language specified in Windows Media Player.

**Syntax**

```
player.settings.defaultAudioLanguage
```

**Possible Values**

This property is a read-only **Number (long)**.

**Remarks**

An LCID uniquely identifies a particular language dialect, called a locale.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also

- [Player.ScriptCommand Event](#)
- [Settings Object](#)
**Settings.defaultFrame**

The `defaultFrame` property specifies or retrieves the name of the frame used to display a URL that is received in a `ScriptCommand` event.

**Syntax**

```
player.settings.defaultFrame
```

**Possible Values**

This property is a read/write `String` corresponding to the value of the `name` attribute of the target `FRAME` element.

**Remarks**

If a target frame is specified in the `ScriptCommand` event itself, this property is ignored.

This property is ignored when using the Netscape Navigator Java applet. In Navigator, each URL-type script command received displays the URL in a new browser window, regardless of the value of `Settings.defaultFrame`.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Player.ScriptCommand Event](#)
- [Settings Object](#)
- [Using Windows Media Player with Netscape Navigator](#)
Settings.enableErrorDialogs

The `enableErrorDialogs` property specifies or retrieves a value indicating whether error dialog boxes are shown automatically.

**Syntax**

```
player.settings.enableErrorDialogs
```

**Possible Values**

This property is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Error dialogs are shown automatically.</td>
</tr>
<tr>
<td>false</td>
<td>Error dialogs are not shown automatically.</td>
</tr>
</tbody>
</table>

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**Remarks**

This property exhibits specific behavior for remoted instances of the Player control. For more information, see [Remoting the Windows Media Player Control](#).

**See Also**

- [Settings Object](#)
Settings.getMode

The `getMode` method determines whether the loop mode or shuffle mode is active.

**Syntax**

```
player.settings.getMode(modeName)
```

**Parameters**

- `modeName` (String) specifying the name of the mode in question, containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoRewind</td>
<td>Mode indicating that the tracks are rewound to the beginning after playing to the end.</td>
</tr>
<tr>
<td>loop</td>
<td>Mode indicating that the sequence of tracks repeats itself.</td>
</tr>
<tr>
<td>showFrame</td>
<td>Mode indicating that the nearest key frame is displayed at the current position when not playing. This mode is not relevant for audio tracks.</td>
</tr>
<tr>
<td>shuffle</td>
<td>Mode indicating that the tracks are played in random order.</td>
</tr>
</tbody>
</table>

**Return Values**

This method returns a `Boolean` value indicating whether the specified mode is active.

**Requirements**

- **Version**: For loop and shuffle modes, Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

For autoRewind and showFrame modes, Windows Media Player 9 Series or later.

**See Also**

- [Settings Object](#)
- [Settings.setMode](#)
Settings.invokeURLs

The invokeURLs property specifies or retrieves a value indicating whether URL events should launch a Web browser.

Syntax

player.settings.invokeURLs

Possible Values

This property is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. URL events should launch a browser.</td>
</tr>
<tr>
<td>false</td>
<td>URL events should not launch a browser.</td>
</tr>
</tbody>
</table>

Remarks

Media files can contain URLs. When a URL is sent to the Windows Media Player control, it is passed first to the ScriptCommand event handler regardless of the value in invokeURLs. After ScriptCommand exits, Windows Media Player checks invokeURLs to determine whether to launch the default Internet browser with the URL. You can selectively display URLs by checking them in ScriptCommand and setting invokeURLs as desired.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Player.ScriptCommand Event
- Settings Object
Settings.isAvailable

The isAvailable property indicates whether a specified type of information is available or a specified action can be performed.

Syntax

player.settings.isAvailable(name)

Parameters

name

String containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoStart</td>
<td>Determines whether the autoStart property can be set.</td>
</tr>
<tr>
<td>Balance</td>
<td>Determines whether the balance property can be set.</td>
</tr>
<tr>
<td>BaseURL</td>
<td>Determines whether the baseURL property can be set.</td>
</tr>
<tr>
<td>DefaultFrame</td>
<td>Determines whether the defaultFrame property can be set.</td>
</tr>
<tr>
<td>EnableErrorDialogs</td>
<td>Determines whether the enableErrorDialogs property can be set.</td>
</tr>
<tr>
<td>GetMode</td>
<td>Determines whether the getMode method can be called.</td>
</tr>
<tr>
<td>InvokeURLs</td>
<td>Determines whether the invokeURLs property can be set.</td>
</tr>
<tr>
<td>Mute</td>
<td>Determines whether the mute property can be set.</td>
</tr>
<tr>
<td>PlayCount</td>
<td>Determines whether the playCount property can be set.</td>
</tr>
<tr>
<td>Rate</td>
<td>Determines whether the rate property can be set.</td>
</tr>
<tr>
<td>SetMode</td>
<td>Determines whether the setMode method can be called.</td>
</tr>
<tr>
<td>Volume</td>
<td>Determines whether the volume property can be set.</td>
</tr>
</tbody>
</table>

Return Values

© 2000-2003 Microsoft Corporation. All rights reserved.
This method returns a **Boolean** value.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

**See Also**

- [Settings Object](#)

---

**Settings.mediaAccessRights**

The **mediaAccessRights** property retrieves a value indicating the rights currently granted for **Media Library** access.

**Syntax**

`player.settings.mediaAccessRights`

**Possible Values**

This property is a read-only **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Current item access rights only.</td>
</tr>
<tr>
<td>read</td>
<td>Read access rights only.</td>
</tr>
<tr>
<td>full</td>
<td>Read/Write access rights.</td>
</tr>
</tbody>
</table>

**Remarks**

A Web page must first request permission from the user to read information from or write data to **Media Library**. This means that certain methods, properties, and events will be inaccessible from code if the appropriate access rights have not been granted. To obtain access rights, the application calls
Settings.requestMediaAccessRights, passing a parameter that specifies the desired access rights level.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Settings Object
- Settings.requestMediaAccessRights

© 2000-2003 Microsoft Corporation. All rights reserved.

Settings.mute

The mute property specifies and retrieves a value indicating whether audio is muted.

Syntax

player.settings.mute

Possible Values

This property is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Audio is muted.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Audio is not muted.</td>
</tr>
</tbody>
</table>

Example Code

The following example creates an HTML CHECKBOX element that allows the user to mute and un-mute audio. The Player object was created with ID = "Player".

```html
<!-- Create an HTML CHECKBOX control. -->
<input type = "CHECKBOX" id = MUTE
```
onClick = "
/* Use the CHECKBOX state to set
the mute property. */
Player.settings.mute = MUTE.checked;
"

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Settings Object

Settings.playCount

The **playCount** property specifies or retrieves the number of times a media item will play.

Syntax

```
player.settings.playCount
```

Possible Values

This property is a read/write **Number (long)** with a minimum value of one and a default value of one.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also
The rate property specifies or retrieves the current playback rate of video media.

Syntax

`player.settings.rate`

Possible Values

This property is a read/write `Number (double)` with a default value of 1.0.

Remarks

This property acts as a multiplier value that allows you to play a clip at a faster or slower rate. The default value of 1.0 indicates the authored speed. Note that an audio track becomes difficult to understand at rates lower than 0.5 or higher than 1.5. A playback rate of 2 equates to twice the normal playback speed.

Windows Media Player will attempt to use the most effective of four different playback modes. These modes are smooth video playback with audio pitch maintained, smooth video playback with audio pitch not maintained, smooth video playback with no audio, and keyframe video playback with no audio. The mode chosen by the Player depends on numerous factors including file type and location, operating system, network, and server.

Other considerations apply as well, depending on media type:

- Windows Media Format (WMV) and ASF files: Optimal values for this property are from 1 to 10, or from –1 to –10 for reverse play. Values from 0.5 to 1.0 or from -0.5 to -1.0 may also work well in cases where audio pitch can be maintained, for example, when playing files located on the local computer. Values with an absolute magnitude greater than 10 are allowed, but are not very meaningful.
- Other Video Media Types: This property can range from 0 to 9. Negative values are not allowed. Values less than 1 represent slow motion. Values above 9 are allowed, but are not very meaningful.

The `Controls.fastForward` method changes the value of `rate` to 5.0, while the `Controls.fastReverse` method changes `rate` to –5.0.

The playback rate of some media types cannot be altered. Use the `Settings.isAvailable` method to determine whether this property can be specified for a particular media item.
Example Code

The following example creates an HTML SELECT element that allows the user to change the playback speed of the current media. The SELECT options offer normal speed, half-speed and double-speed playback rates. The Player object was created with ID = "Player".

<!-- Create the HTML SELECT element. -->
<SELECT ID = pbRATE NAME = "pbRATE" LANGUAGE="JScript"
onChange="
    /* Test whether playback rate can be set. */
    if(Player.settings.IsAvailable('Rate'))
        /* Set the playback rate based on the current
        value of the SELECT element. */
        Player.settings.rate = this.value"
/>

/* Create the OPTION list. */
<OPTION VALUE = 1>NORMAL</OPTION>
<OPTION VALUE = .5>half speed</OPTION>
<OPTION VALUE = 2>2 speed</OPTION>
</SELECT>

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Controls.fastForward
- Controls.fastReverse
- Settings Object
- Settings.isAvailable

© 2000-2003 Microsoft Corporation. All rights reserved.

Settings.requestMediaAccessRights

The requestMediaAccessRights method requests a specified level of access to Media Library.
Syntax

player.settings.requestMediaAccessRights\(access\)

Parameters

access

String specifying the desired access rights level. Contains one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Current item access rights only.</td>
</tr>
<tr>
<td>read</td>
<td>Read access rights only.</td>
</tr>
<tr>
<td>full</td>
<td>Read/Write access rights.</td>
</tr>
</tbody>
</table>

Return Values

This method returns a Boolean value indicating whether the requested access rights were granted.

Remarks

A Web page must first request permission from the user to read information from or write data to Media Library. Invoking this method prompts the user with a dialog box that requests the specified permission level. This means that certain methods, properties, and events will be inaccessible from code if the appropriate access rights have not been granted. The current access rights level can be retrieved using Settings.mediaAccessRights.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Settings Object
- Settings.mediaAccessRights
Settings.setMode

The `setMode` method sets various modes to active or inactive.

Syntax

```plaintext
player.settings.setMode(modeName, state)
```

Parameters

- `modeName`  
  String specifying the name of the mode being changed, containing one of the following values.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoRewind</td>
<td>Mode indicating whether the tracks are rewound to the beginning after playing to the end. Default state is true.</td>
</tr>
<tr>
<td>loop</td>
<td>Mode indicating whether the sequence of tracks repeats itself. Default state is false.</td>
</tr>
<tr>
<td>showFrame</td>
<td>Mode indicating whether the nearest video key frame is displayed at the current position when not playing. Default state is false. Has no effect on audio tracks.</td>
</tr>
<tr>
<td>shuffle</td>
<td>Mode indicating whether the tracks are played in random order. Default state is false.</td>
</tr>
</tbody>
</table>

- `state`  
  Boolean specifying the new state of the specified mode.

Return Values

This method does not return a value.

Remarks

When the `showFrame` mode is active, the Player must access the track content to retrieve the video frame. Due to bandwidth considerations, use this mode cautiously when playing non-local content.

Requirements

- **Version**: For loop and shuffle modes, Windows Media Player version 7.0 or later.

- **Header**: Defined in wmp.idl; include wmp.h.
Library: Use wmp.dll.

For autoRewind and showFrame modes, Windows Media Player 9 Series or later.

See Also

- Settings Object
- Settings.getMode

Settings.volume

The volume property specifies or retrieves the current volume.

Syntax

player.settings.volume

Possible Values

This property is a read/write Number (long) ranging from 0 to 100.

Remarks

Zero specifies no volume and 100 specifies full volume. If no value is specified for this property, it defaults to the last volume setting established for the player.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Settings Object
StringCollection Object

The StringCollection object provides a way to manipulate a collection of strings.

The StringCollection object supports the following property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Retrieves the number of items in the string collection.</td>
</tr>
</tbody>
</table>

The StringCollection object supports the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Retrieves the string at the given index.</td>
</tr>
</tbody>
</table>

The StringCollection object is accessed through the following method.

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaCollection</td>
<td>getAttributeStringCollection</td>
</tr>
</tbody>
</table>

For purposes of illustration, `player.mediaCollection.getAttributeStringCollection(attribute, mediaType)` is used in the reference syntax sections.

See Also

- Object Model Reference
StringCollection.count

The `count` property retrieves the number of items in the string collection.

**Syntax**

```javascript
player.mediaCollection.getAttributeStringCollection(attribute, mediaType).count
```

**Possible Values**

This property is a read-only `Number (long)`.

**Remarks**

To retrieve the value of this property, read access to Media Library is required. For more information, see Media Library Access.

**Requirements**

- **Version**: Windows Media Player version 7.0 or later.
- **Header**: Defined in wmp.idl; include wmp.h.
- **Library**: Use wmp.dll.

**See Also**

- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
- StringCollection Object

© 2000-2003 Microsoft Corporation. All rights reserved.

---

StringCollection.item

The `item` method retrieves the string at the given index.

**Syntax**
player.mediaCollection.getAttributeStringCollection(attribute, mediaType).item(index)

Parameters

index

Number (long) containing the index.

Return Values

This method returns a String.

Remarks

The StringCollection object is used to retrieve the set of values available for an attribute. For example, the MediaCollection.getAttributeStringCollection method can be used to retrieve a StringCollection object representing all the values for the Genre attribute within the Audio media type. The item property can then be used to iterate through all of the possible values for the Genre attribute.

To use this method, read access to Media Library is required. For more information, see Media Library Access.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- MediaCollection.getAttributeStringCollection
- Settings.mediaAccessRights
- Settings.requestMediaAccessRights
- StringCollection Object

© 2000-2003 Microsoft Corporation. All rights reserved.
The Windows Media Player Object Model Reference is designed for use with script languages such as Microsoft JScript, but if you are a programming in C++, you can translate the information it provides into C++ terms with minimal effort. Along with some basic translation rules, you'll need to learn the organization and names of the interfaces that the scriptable objects are based on. If you want to use remoting or skins with the Windows Media Player control, you'll also need to learn about the `IWMPPlayerServices` and `IWMPRemoteMediaServices` interfaces, which can only be used with C++.

For a demonstration of embedding the Windows Media Player control in a C++ program, see the WMPHost sample included with the Windows Media Player Software Development Kit (SDK).

C++ information is described in the following sections.

## Section Description

**C++ Translation Guide**

This section provides the translation rules for reading the script reference topics from a C++ perspective.

**Interfaces**

This section provides reference tables that map C++ interface and method names to the corresponding script reference topics. This section also provides complete documentation for interfaces available only through C++.

**Enumeration Types**

This section documents the enumeration types available in C++ code.

### See Also

- [Object Model Reference](#)
- [Embedding the Player Control in a C++ Program](#)

## C++ Translation Guide

This section contains the information you need to translate the script-oriented Object Model Reference into C++ terms. The script reference has almost all the information you need to use the Windows Media Player control in a C++ program, but the syntax may be unfamiliar. Programming the control in C++ requires the use of Component Object Model (COM) conventions, for example. The differences between script and C++ syntax are described for the following areas:

- [Data Type Translation](#)
- [Method Translation](#)
- [Property Translation](#)
Data Type Translation

The script reference refers to three basic data types: **Number**, **String**, and **Boolean**. In the reference, the **Number** keyword is always followed by a more specific type name in parentheses, such as `long`. This is the type used in C++. **String** and **Boolean** values, on the other hand, always use the **BSTR** and **VARIANT_BOOL** types in C++. In addition to the three basic types, **Object** values are also used. These become pointers to COM interfaces in C++, and are described in more detail in a later section.

The following table summarizes these data type conversions.

<table>
<thead>
<tr>
<th>Script type</th>
<th>C++ type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td><code>int</code>, <code>long</code>, <code>double</code>, or <code>float</code> (specific type is indicated in the script reference)</td>
</tr>
<tr>
<td>String</td>
<td><strong>BSTR</strong></td>
</tr>
<tr>
<td>Boolean</td>
<td><strong>VARIANT_BOOL</strong></td>
</tr>
<tr>
<td>Object</td>
<td>Pointer to matching <strong>Interface</strong></td>
</tr>
</tbody>
</table>

A notable exception is the **IWMPPMedia3::getItemInfoByType** method, documented as **Media.getItemInfoByType** in the JScript reference. The C++ return value for this method is type **VARIANT**, because in script it can return different types depending on the context. **ErrorItem.errorContext** also returns type **VARIANT** in C++.

See Also

- **C++ Translation Guide**
- **ErrorItem.errorContext**
- **Media.getItemInfoByType**
Method Translation

Methods in script and methods in C++ differ in the way they return values. All Player methods return HRESULT values in C++, following the COM convention. The return values listed in the script reference are returned using "[out, retval]" parameters in C++. An out parameter is added to the end of the documented list of regular (or "[in]"") parameters. An out parameter is always a pointer to the return type. This pointer is declared and allocated, then passed to a method when the method is called. The method assigns the return value to the pointer. The calling routine accesses the value through the pointer, then frees the pointer resources when it is finished.

For example, in Microsoft JScript, the following syntax block is for a method call that returns a String containing the value of the attribute with the specified name (which is also a String value):

player.currentMedia.getItemInfo(name)

The C++ equivalent returns an HRESULT, and places the value of the requested attribute at the location specified by the pbstrVal pointer:

HRESULT getItemInfo(BSTR bstrItemName, BSTR* pbstrVal);

For more information on Player method signatures for C++, see the wmp.idl file shipped with the Windows Media Player Software Development Kit (SDK).

See Also

- C++ Translation Guide
All the properties documented in the script reference are specified or retrieved using special methods in C++ called accessor methods. The names of these methods are obtained by adding **get**_ or **put**_ to the front of the documented property names. If a property is read/write, there will be both **get**_ and **put**_ methods. If a property is read-only, there will be a **get**_ method only.

Aside from their use to specify or retrieve property values, accessor methods are just like other methods. As described earlier, the return value is always an **HRESULT**. The value that is specified or retrieved is an in or an out parameter, respectively. For example, in JScript, the following syntax block is for a read-only property that contains a **String** value representing the duration of the current media item:

```
player.currentMedia.durationString
```

The C++ equivalent is an accessor method that returns an **HRESULT** and places the **durationString** value at the location specified by the **pbstrDuration** pointer:

```
HRESULT get_durationString(BSTR* pbstrDuration);
```

See Also

- **C++ Translation Guide**

---

**Object Translation**

An **Object** in the script reference corresponds to an **Interface** pointer in C++. The Player interface names begin with **IWMP** followed by the object name documented in the script reference. Some objects in the script reference, however, correspond to multiple interfaces. If an interface has been updated, for example, it will have multiple, sequentially numbered versions. Additionally, the **Player** object corresponds to the **IWMPCore**, **IWMPPlayer**, and **_WMPOCXEvents** interfaces, the first two of which also have multiple versions.

When an interface is an update of another interface, its name ends with a number indicating the version, for example **IWMPMedia2**, which supplements **IWMPMedia**. When you retrieve a pointer to a base interface (the first version, which doesn't have a version number) you must call **QueryInterface** on it in order to retrieve a pointer to the version that includes the method you want to call. For example, when you want to call **get_currentPositionTimecode**, you must first retrieve an **IWMPControls** pointer by calling **get_controls** through the **IWMPCore** interface. Next, retrieve an **IWMPControls3** pointer by calling **QueryInterface** on the **IWMPControls** pointer. If the **IWMPControls3** pointer is retrieved successfully, it is then used to call **get_currentPositionTimecode**.

This is illustrated in the following code, in which spCore is of type **CComPtr<IWMPCore>**.
CComPtr<IWMPControls> spControls;
CComPtr<IWMPControls3> spControls3;
CComBSTR bstrTimecode;
HRESULT hr = S_OK;
if ( SUCCEEDED(hr) )
{
    hr = spCore->get_controls(&spControls);
}
if ( SUCCEEDED(hr) )
{
    hr = spControls->QueryInterface(&spControls3);
}
if ( SUCCEEDED(hr) )
{
    hr = spControls3->get_currentPositionTimecode( &bstrTimecode );
}
if ( SUCCEEDED(hr) )
{
    MessageBox( bstrTimecode );
}

When the script reference refers to an Object without a specific type, the exact type can be found in the wmp.idl file shipped with the SDK in the include folder.

See Also

- C++ Translation Guide

## Interfaces

This section lists the names of the COM interfaces exposed by the Windows Media Player control. The accessor methods of the IWMPCore interface are used to retrieve specific interfaces. These interfaces, in turn, may have accessor methods for retrieving further interfaces. Calling QueryInterface on one of these interfaces is only necessary when you retrieve the base version of an interface and wish to call a method on a later version of the same interface.

A scriptable object incorporates all versions of its corresponding interface, but in C++, you must have either a pointer to the version that first includes the methods you need or a pointer to any later version.

Clicking the interface name will take you to a table listing the methods supported by that version. The description of each method contains a link that will take you to the corresponding script reference section.

The Windows Media Player control supports the following interfaces.
<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_WMPOCXEvents</td>
<td>Events originating from the Windows Media Player control to which an embedd</td>
</tr>
<tr>
<td>IWMPCdrom</td>
<td>Methods for accessing a CD or DVD disc in its drive.</td>
</tr>
<tr>
<td>IWMPCdromCollection</td>
<td>Methods for accessing a collection of CD-ROM or DVD-ROM drives.</td>
</tr>
<tr>
<td>IWMPClosedCaption</td>
<td>Methods for including captions with a media clip.</td>
</tr>
<tr>
<td>IWMPClosedCaption2</td>
<td>Additional closed-captioning methods.</td>
</tr>
<tr>
<td>IWMPControls</td>
<td>Methods representing the transport controls of Windows Media Player, such</td>
</tr>
<tr>
<td>IWMPControls2</td>
<td>Additional control methods.</td>
</tr>
<tr>
<td>IWMPControls3</td>
<td>Additional control methods.</td>
</tr>
<tr>
<td>IWMPCore</td>
<td>Methods used to retrieve pointers to other interfaces and to access basic</td>
</tr>
<tr>
<td>IWMPCore2</td>
<td>Additional core methods.</td>
</tr>
<tr>
<td>IWMPCore3</td>
<td>Additional core methods.</td>
</tr>
<tr>
<td>IWMPDVD</td>
<td>Methods for working with DVDs.</td>
</tr>
<tr>
<td>IWMPError</td>
<td>Methods providing access to a collection of IWMPErrorItem pointers.</td>
</tr>
<tr>
<td>IWMPErrorItem</td>
<td>Methods that provide information about errors.</td>
</tr>
<tr>
<td>IWMPErrorItem2</td>
<td>Additional error item methods.</td>
</tr>
<tr>
<td>IWMPEvents</td>
<td>Events originating from the Windows Media Player control to which an embedd</td>
</tr>
<tr>
<td>IWMPMedia</td>
<td>Methods relating to multimedia clips.</td>
</tr>
<tr>
<td>IWMPMedia2</td>
<td>Additional media methods.</td>
</tr>
<tr>
<td>IWMPMedia3</td>
<td>Additional media methods.</td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>Methods for accessing a collection of IWMPMedia pointers.</td>
</tr>
<tr>
<td>IWMPMetadataPicture</td>
<td>Methods for retrieving information about the WM/Picture metadata attribute.</td>
</tr>
<tr>
<td>IWMPMetadataText</td>
<td>Methods for retrieving information about complex textual metadata attributes.</td>
</tr>
<tr>
<td>IWMPNetwork</td>
<td>Methods relating to the network connection of Windows Media Player.</td>
</tr>
<tr>
<td>IWMPPlayer</td>
<td>Methods for modifying the basic behavior of the control user interface.</td>
</tr>
</tbody>
</table>
### Object Model Reference for C++

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPPlayer2</td>
<td>Additional Player methods.</td>
</tr>
<tr>
<td>IWMPPlayer3</td>
<td>Additional Player methods.</td>
</tr>
<tr>
<td>IWMPPlayer4</td>
<td>Additional Player methods.</td>
</tr>
<tr>
<td>IWMPPlayerApplication</td>
<td>Methods for switching between a remoted Player control and the full mode of the Player. Can only be used with C++ programs that embed the control in remote mode.</td>
</tr>
<tr>
<td>IWMPPlayerServices</td>
<td>Methods used by the host of a remoted control to manipulate the full mode of the Player. Can only be used with C++.</td>
</tr>
<tr>
<td>IWMPPlaylist</td>
<td>Methods for manipulating lists of media clips.</td>
</tr>
<tr>
<td>IWMPPlaylistArray</td>
<td>Methods for accessing a collection of IWMPPlaylist pointers by index number.</td>
</tr>
<tr>
<td>IWMPPlaylistCollection</td>
<td>Methods for manipulating IWMPPlaylist and IWMPPlaylistArray pointers.</td>
</tr>
<tr>
<td>IWMPRemoteMediaServices</td>
<td>Methods that provide services to the Player from a program that hosts the Player control. Can only be used with C++.</td>
</tr>
<tr>
<td>IWMPSettings</td>
<td>Methods that allow the specification or retrieval of Windows Media Player settings.</td>
</tr>
<tr>
<td>IWMPSettings2</td>
<td>Additional settings methods.</td>
</tr>
<tr>
<td>IWMPSkinManager</td>
<td>A method for specifying the skin used with Windows Media Player.</td>
</tr>
<tr>
<td>IWMPStringCollection</td>
<td>Methods for accessing a collection of strings.</td>
</tr>
</tbody>
</table>

See Also

- [Object Model Reference for C++](#)
The _WMPOCXEvents interface provides events originating from the Windows Media Player control to which an embedding program can respond.

In addition to the methods inherited from IDispatch, the _WMPOCXEvents interface exposes the following events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AudioLanguageChange</td>
<td>Occurs when the current audio language changes. See Player.AudioLanguageChange Event.</td>
</tr>
<tr>
<td>Buffering</td>
<td>Occurs when the Windows Media Player control begins or ends buffering. See Player.Buffering Event.</td>
</tr>
<tr>
<td>CdromMediaChange</td>
<td>Occurs when a CD or DVD is inserted into or ejected from a CD-ROM or DVD-ROM drive. See Player.CdromMediaChange Event.</td>
</tr>
<tr>
<td>Click</td>
<td>Occurs when the user clicks a mouse button. See Player.Click Event.</td>
</tr>
<tr>
<td>CurrentItemChange</td>
<td>Occurs when the currentItem property of the IWMPControls interface changes value. See Player.CurrentItemChange Event.</td>
</tr>
<tr>
<td>CurrentMediaItemAvailable</td>
<td>Occurs when the current media item becomes available. See Player.CurrentMediaItemAvailable Event.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>DomainChange</td>
<td>Occurs when the DVD domain changes. See Player.DomainChange Event.</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>Occurs when the user double-clicks a mouse button. See Player.DoubleClick Event.</td>
</tr>
<tr>
<td>DurationUnitChange</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>EndOfStream</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Error</td>
<td>Occurs when the Windows Media Player control has an error condition. See Player.Error Event.</td>
</tr>
<tr>
<td>KeyDown</td>
<td>Occurs when a key is pressed. See Player.KeyDown Event.</td>
</tr>
<tr>
<td>KeyPress</td>
<td>Occurs when a key is pressed and then released. See Player.KeyPress Event.</td>
</tr>
<tr>
<td>KeyUp</td>
<td>Occurs when a key is released. See Player.KeyUp Event.</td>
</tr>
<tr>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>MarkerHit</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a marker is reached. See <strong>Player.MarkerHit</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a media item changes. See <strong>Player.MediaChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaCollectionAttributeStringAdded</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when an attribute is added to <strong>Media Library</strong>. See <strong>Player.MediaCollectionAttributeStringAdded</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaCollectionAttributeStringChanged</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when an attribute in <strong>Media Library</strong> is changed. See <strong>Player.MediaCollectionAttributeStringChanged</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaCollectionAttributeStringRemoved</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when an attribute is removed from <strong>Media Library</strong>. See <strong>Player.MediaCollectionAttributeStringRemoved</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaCollectionChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when the media collection changes. See <strong>Player.MediaCollectionChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MediaError</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when the media object has an error condition. See <strong>Player.MediaError</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>ModeChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a mode of the player is changed. See <strong>Player.ModeChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MouseDown</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a mouse button is pressed. See <strong>Player.MouseDown</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MouseMove</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when the mouse pointer is moved. See <strong>Player.MouseMove</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>MouseUp</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a mouse button is released. See <strong>Player.MouseUp</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>NewStream</strong></td>
<td></td>
</tr>
<tr>
<td>Reserved for future use.</td>
<td></td>
</tr>
<tr>
<td><strong>OpenPlaylistSwitch</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a title on a DVD begins playing. See <strong>Player.OpenPlaylistSwitch</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>OpenStateChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when the Windows Media Player control changes state. See <strong>Player.OpenStateChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>PlayerDockedStateChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a remoted Player control docks or undocks. See <strong>Player.PlayerDockedStateChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>PlaylistChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when a playlist changes. See <strong>Player.PlaylistChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td><strong>PlaylistCollectionChange</strong></td>
<td></td>
</tr>
<tr>
<td>Occurs when something changes in the playlist collection. See <strong>Player.PlaylistCollectionChange</strong> Event.</td>
<td></td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>PlaylistCollectionPlaylistAdded</strong></td>
<td>Occurs when a playlist is added to the playlist collection. See <strong>Player.PlaylistCollectionPlaylistAdded</strong> Event.</td>
</tr>
<tr>
<td><strong>PlaylistCollectionPlaylistRemoved</strong></td>
<td>Occurs when a playlist is removed from the playlist collection. See <strong>Player.PlaylistCollectionPlaylistRemoved</strong> Event.</td>
</tr>
<tr>
<td><strong>PlaylistCollectionPlaylistSetAsDeleted</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>PlayStateChange</strong></td>
<td>Occurs when the play state of the Windows Media Player control changes. See <strong>Player.PlayStateChange</strong> Event.</td>
</tr>
<tr>
<td><strong>PositionChange</strong></td>
<td>Occurs when the current position of the media has been changed. See <strong>Player.PositionChange</strong> Event.</td>
</tr>
<tr>
<td><strong>ScriptCommand</strong></td>
<td>Occurs when a synchronized command or URL is received. See <strong>Player.ScriptCommand</strong> Event.</td>
</tr>
<tr>
<td><strong>StatusChange</strong></td>
<td>Occurs when the <strong>status</strong> property changes value. See <strong>Player.StatusChange</strong> Event.</td>
</tr>
<tr>
<td><strong>SwitchedToControl</strong></td>
<td>Occurs when a remoted Player control switches to the docked state. See <strong>Player.SwitchedToControl</strong> Event.</td>
</tr>
<tr>
<td><strong>SwitchedToPlayerApplication</strong></td>
<td>Occurs when a remoted Player control switches to the full mode of the Player. See <strong>Player.SwitchedToPlayerApplication</strong> Event.</td>
</tr>
<tr>
<td><strong>Warning</strong></td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

See Also

- C++ Translation Guide
- Interfaces

---

The **IWMPCdrom** interface provides methods that access a CD or DVD disc in its drive.

In addition to the methods inherited from **IDispatch**, the **IWMPCdrom** interface exposes the following
methods.

**Method** | **Description**
--- | ---
eaject | Eject the disc from the drive. See `Cdrom.eject`.
<get_driveSpecifier> | Retrieves the CD-ROM or DVD-ROM drive letter. See `Cdrom.driveSpecifier`.
<get_playlist> | Retrieves a pointer to an `IWMPPplaylist` interface representing the tracks on the CD disc or the titles on the DVD disc currently in the drive. See `Cdrom.playlist`.

Retrieve a pointer to an `IWMPCdrom` interface with the following method.

**Interface** | **Method**
--- | ---
`IWMPCdromCollection` | item - see `CdromCollection.item`

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

**See Also**
- [C++ Translation Guide](#)
- [Interfaces](#)

---

**IWMPCdromCollection Interface**

The `IWMPCdromCollection` interface provides methods for accessing a collection of CD-ROM or DVD-ROM drives.

In addition to the methods inherited from `IDispatch`, the `IWMPCdromCollection` interface exposes the following methods.

**Method** | **Description**
--- | ---
item | Retrieves a pointer to the `IWMPCdrom` interface specified by index. See `CdromCollection.item`.

© 2000-2003 Microsoft Corporation. All rights reserved.
get_count

Retrieves the number of CD-ROM and DVD-ROM drives on the system. See `CdromCollection.count`.

getByDriveSpecifier

Retrieves a pointer to the `IWMPCdrom` interface associated with the specified drive letter. See `CdromCollection.getByDriveSpecifier`.

Retrieve a pointer to an `IWMPCdromCollection` interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCore</td>
<td><code>get_cdromCollection</code> - see <code>Player.cdromCollection</code></td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- [C++ Translation Guide](#)
- [Interfaces](#)

---

IWMPClosedCaption Interface

The `IWMPClosedCaption` interface provides a way to include captions with a media clip.

In addition to the methods inherited from `IDispatch`, the `IWMPClosedCaption` interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_captioningID</code></td>
<td>Retrieves the name of the frame or control displaying the captioning. See <code>ClosedCaption.captioningID</code>.</td>
</tr>
<tr>
<td><code>get_SAMIFileName</code></td>
<td>Retrieves the name of the file that contains information needed for closed captioning. See <code>ClosedCaption.SAMIFileName</code>.</td>
</tr>
<tr>
<td><code>get_SAMILang</code></td>
<td>Retrieves the language displayed for closed captioning. See <code>ClosedCaption.SAMILang</code>.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an `IWMPClosedCaption` interface with the following method.

### Interface Method

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCore</td>
<td>get_closedCaption. See Player.closedCaption</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
- `IWMPClosedCaption2` Interface

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPClosedCaption2 Interface**

The `IWMPClosedCaption2` interface provides closed captioning methods that supplement the `IWMPClosedCaption` interface.

In addition to the methods inherited from `IWMPClosedCaption`, the `IWMPClosedCaption2` interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_SAMIStyle</td>
<td>Retrieves the closed-captioning style. See <code>ClosedCaption.SAMIStyle</code>.</td>
</tr>
<tr>
<td>put_captioningID</td>
<td>Specifies the name of the frame or control displaying the captioning. See <code>ClosedCaption.captioningID</code>.</td>
</tr>
<tr>
<td>put_SAMIFileName</td>
<td>Specifies the name of the file that contains the information needed for closed captioning. See <code>ClosedCaption.SAMIFileName</code>.</td>
</tr>
<tr>
<td>put_SAMILang</td>
<td>Specifies the language displayed for closed captioning. See <code>ClosedCaption.SAMILang</code>.</td>
</tr>
<tr>
<td>put_SAMIStyle</td>
<td>Specifies the closed-captioning style. See <code>ClosedCaption.SAMIStyle</code>.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an `IWMPClosedCaption2` interface by calling the `QueryInterface` method of an `IWMPClosedCaption` interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
- `IWMPClosedCaption` Interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_SAMILangCount</code></td>
<td>Retrieves the number of languages supported by the current SAMI file. See <code>ClosedCaption.SAMILangCount</code>.</td>
</tr>
<tr>
<td><code>getSAMILangID</code></td>
<td>Retrieves the locale identifier (LCID) of a language supported by the current SAMI file. See <code>ClosedCaption.getSAMILangID</code>.</td>
</tr>
<tr>
<td><code>getSAMILangName</code></td>
<td>Retrieves the name of a language supported by the current SAMI file. See <code>ClosedCaption.getSAMILangName</code>.</td>
</tr>
<tr>
<td><code>get_SAMIStyleCount</code></td>
<td>Retrieves the number of styles supported by the current SAMI file. See <code>ClosedCaption.SAMIStyleCount</code>.</td>
</tr>
<tr>
<td><code>getSAMIStyleName</code></td>
<td>Retrieves the name of a style supported by the current SAMI file. See <code>ClosedCaption.getSAMIStyleName</code>.</td>
</tr>
</tbody>
</table>

© 2000-2003 Microsoft Corporation. All rights reserved.
Retrieve a pointer to an **IWMPCore** interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
</table>
| **IWMPCore**   | *get_controls* | See **Player.controls**.

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.
IWMPControls2 Interface

The **IWMPControls2** interface provides a method that supplements the **IWMPControls** interface.

In addition to the methods inherited from **IWMPControls**, the **IWMPControls2** interface exposes the following method:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>step</td>
<td>Stops playback of the current DVD media item on the next frame. See <strong>Controls.step</strong>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPControls2** interface by calling the **QueryInterface** method of an **IWMPControls** interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- **C++ Translation Guide**
- **Interfaces**
- **IWMPControls Interface**
- **IWMPControls3 Interface**
# IWMPControls3 Interface

The **IWMPControls3** interface provides methods that supplement the **IWMPControls2** interface.

In addition to the methods inherited from **IWMPControls2**, the **IWMPControls3** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_audioLanguageCount</td>
<td>Retrieves the count of supported audio languages. See <strong>Controls.audioLanguageCount</strong></td>
</tr>
<tr>
<td>get_currentAudioLanguage</td>
<td>Retrieves the locale identifier (LCID) of the audio language for playback. See <strong>Controls.currentAudioLanguage</strong></td>
</tr>
<tr>
<td>get_currentAudioLanguageIndex</td>
<td>Retrieves the index that corresponds to the audio language for playback. See <strong>Controls.currentAudioLanguageIndex</strong></td>
</tr>
<tr>
<td>get_currentPositionTimecode</td>
<td>Retrieves the current position in the current media using a time code format. This property supports SMPTE time code. See <strong>Controls.currentPositionTimecode</strong></td>
</tr>
<tr>
<td>getAudioLanguageDescription</td>
<td>Retrieves the description for the audio language corresponding to the specified index. See <strong>Controls.getAudioLanguageDescription</strong></td>
</tr>
<tr>
<td>getAudioLanguageID</td>
<td>Retrieves the LCID for a specified audio language index. See <strong>Controls.getAudioLanguageID</strong></td>
</tr>
<tr>
<td>getLanguageName</td>
<td>Retrieves the name of the audio language with the specified LCID. See <strong>Controls.getLanguageName</strong></td>
</tr>
<tr>
<td>put_currentAudioLanguage</td>
<td>Specifies the LCID of the audio language for playback. See <strong>Controls.currentAudioLanguage</strong></td>
</tr>
<tr>
<td>put_currentAudioLanguageIndex</td>
<td>Specifies the index that corresponds to the audio language for playback. See <strong>Controls.currentAudioLanguageIndex</strong></td>
</tr>
<tr>
<td>put_currentPositionTimecode</td>
<td>Specifies the current position in the current media using SMPTE time code format. See <strong>Controls.currentPositionTimecode</strong></td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPControls3** interface by calling the **QueryInterface** method of an **IWMPControls** interface.
IWMPCore Interface

The IWMPCore interface is the root interface for the Windows Media Player control. It can be used to retrieve pointers to other interfaces supported by the control and to access some basic features.

In addition to the methods inherited from IDispatch, the IWMPCore interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>Closes Windows Media Player. See Player.close.</td>
</tr>
<tr>
<td>get_cdromCollection</td>
<td>Retrieves an IWMPCdromCollection pointer. See Player.cdromCollection.</td>
</tr>
<tr>
<td>get_closedCaption</td>
<td>Retrieves an IWMPClosedCaption pointer. See Player.closedCaption.</td>
</tr>
<tr>
<td>get_controls</td>
<td>Retrieves an IWMPControls pointer. See Player.controls.</td>
</tr>
<tr>
<td>get_currentMedia</td>
<td>Retrieves an IWMPMedia pointer corresponding to the current media item. See Player.currentMedia.</td>
</tr>
<tr>
<td>get_currentPlaylist</td>
<td>Retrieves an IWMPPplaylist pointer corresponding to the current playlist. See Player.currentPlaylist.</td>
</tr>
<tr>
<td>get_error</td>
<td>Retrieves an IWMPError pointer. See Player.error.</td>
</tr>
<tr>
<td>get_isOnline</td>
<td>Retrieves a value indicating whether the user is connected to a network. See Player.isOnline.</td>
</tr>
<tr>
<td>get_mediaCollection</td>
<td>Retrieves an IWMPMediaCollection pointer. See Player.mediaCollection.</td>
</tr>
</tbody>
</table>
The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for `IWMPPlayer4`, which inherits from this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPCore2 Interface
- IWMPCore3 Interface

get_network

Retrieves an `IWMPNetwork` pointer. See `Player.network`.

get_openState

Retrieves a value indicating the state of the content source. See `Player.openState`.

get_playlistCollection

Retrieves an `IWMPPlaylistCollection` pointer. See `Player.playlistCollection`.

get_playState

Retrieves a value indicating the operating state of Windows Media Player. See `Player.playState`.

get_settings

Retrieves an `IWMPSettings` pointer. See `Player.settings`.

get_status

Retrieves a value indicating the current status of Windows Media Player. See `Player.status`.

get_URL

Retrieves the name of the clip to play. See `Player.URL`.

get_versionInfo

Retrieves a `String` value specifying the version of Windows Media Player. See `Player.versionInfo`.

launchURL

Sends a URL to the user's default browser. See `Player.launchURL`.

put_currentMedia

Specifies the `IWMPMedia` pointer that corresponds to the current media item. See `Player.currentMedia`.

put_currentPlaylist

Specifies the `IWMPPlaylist` pointer that corresponds to the current playlist. See `Player.currentPlaylist`.

put_URL

Specifies the name of the clip to play. See `Player.URL`.
IWMPCore2 Interface

The IWMPCore2 interface provides a method that supplements the IWMPCore interface.

In addition to the methods inherited from IWMPCore, the IWMPCore2 interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_dvd</td>
<td>Retrieves an IWMPDVD pointer. See Player.dvd.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an IWMPCore2 interface by calling the QueryInterface method of the IWMPCore interface.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for IWMPPlayer4, which inherits from this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPCore Interface
- IWMPCore3 Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPCore3 Interface

The IWMPCore3 interface provides methods that supplement the IWMPCore2 interface.

In addition to the methods inherited from IWMPCore2, the IWMPCore3 interface exposes the following methods.
Retrieve a pointer to an IWMPCore3 interface by calling the QueryInterface method of the IWMPCore or IWMPCore2 interfaces.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for IWMPPlayer4, which inherits from this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPCore Interface
- IWMPCore2 Interface

IWMPDVD Interface

The IWMPDVD interface provides methods for working with DVDs.

In addition to the methods inherited from IDispatch, the IWMPDVD interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newMedia</td>
<td>Retrieves an IWMPMedia pointer for a new media item. See Player.newMedia.</td>
</tr>
<tr>
<td>newPlaylist</td>
<td>Retrieves an IWMPPlaylist pointer for a new playlist. See Player.newPlaylist.</td>
</tr>
<tr>
<td>back</td>
<td>Changes the display from a submenu to its parent menu. See DVD.back.</td>
</tr>
<tr>
<td>get_domain</td>
<td>Retrieves the current domain of the DVD. See DVD.domain.</td>
</tr>
<tr>
<td>get_isAvailable</td>
<td>Retrieves whether a specified type of information is available or a given action can be performed. See DVD.isAvailable.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an IWMPDVD interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCore2</td>
<td>get_dvd - see Player.dvd</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces

IWMPError Interface

The IWMPError interface provides methods for accessing a collection of IWMPErrorItem pointers.

In addition to the methods inherited from IDispatch, the IWMPError interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearErrorQueue</td>
<td>Clears the errors from the error queue. See Error.clearErrorQueue.</td>
</tr>
<tr>
<td>get_errorCount</td>
<td>Retrieves the number of errors in the error queue. See Error.errorCount.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves a pointer to an IWMPErrorItem interface</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an **IWMPErr** interface with the following method.

### Interface Method

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IWMPCore</strong></td>
<td><strong>get_error</strong></td>
<td>See <strong>Player.error</strong>.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

**See Also**
- *C++ Translation Guide*
- *Interfaces*

---

### IWMPErrorItem Interface

The **IWMPErrorItem** interface provides a way to access error information.

In addition to the methods inherited from **IDispatch**, the **IWMPErrorItem** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get_customURL</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>get_errorCode</strong></td>
<td>Retrieves the current error code. See <strong>ErrorItem.errorCode</strong>.</td>
</tr>
<tr>
<td><strong>get_errorContext</strong></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><strong>get_errorDescription</strong></td>
<td>Retrieves a description of the error. See <strong>ErrorItem.errorDescription</strong>.</td>
</tr>
<tr>
<td><strong>get_remedy</strong></td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an **IWMPErrorItem** interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IWMPError</strong></td>
<td>get_item</td>
<td>See <strong>Error.item</strong>.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)
- [IWMPErrorItem2 Interface](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPErrorItem2 Interface**

The **IWMPErrorItem2** interface provides a method that supplements the **IWMPErrorItem** interface.

In addition to the methods inherited from **IWMPErrorItem**, the **IWMPErrorItem2** interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get_condition</strong></td>
<td>Retrieves a value indicating the condition for the error. See <strong>ErrorItem.condition</strong>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPErrorItem2** interface by calling the **QueryInterface** method of the **IWMPErrorItem** interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)
- [IWMPErrorItem Interface](#)
IWMPEvents Interface

The **IWMPEvents** interface provides the same events as the **_WMPOCXEvents** interface, but inherits from **IUnknown** rather than **IDispatch**. This allows you to receive Windows Media Player event notifications directly.

The Windows Media Player Plug-in Wizard can create a sample project that demonstrates how to implement the **IWMPEvents** interface. For details about how to create a sample UI plug-in project that includes event handling, see [Building a UI Plug-in](#).

See Also

- **_WMPOCXEvents** Interface
- C++ Translation Guide
- Interfaces

IWMPMedia Interface

The **IWMPMedia** interface provides methods for specifying and retrieving properties of a multimedia clip.

In addition to the methods inherited from **IDispatch**, the **IWMPMedia** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_attributeCount</td>
<td>Retrieves the number of attributes that can be queried and/or set for the media item. See <a href="#">Media.attributeCount</a>.</td>
</tr>
</tbody>
</table>
get_duration

Retrieves the duration (in seconds) of the current media item. See Media.duration.

get_durationString

Retrieves a String value indicating the duration of the current media item in HH:MM:SS format. See Media.durationString.

get_imageSourceHeight

Retrieves the height of the current media item in pixels. See Media.imageSourceHeight.

get_imageSourceWidth

Retrieves the width of the current media item in pixels. See Media.imageSourceWidth.

get_isIdentical

Retrieves a value indicating whether the supplied object is the same as the current one. See Media.isIdentical.

get_markerCount

Retrieves the number of markers in the media item. See Media.markerCount.

get_name

Retrieves the name of the media item. See Media.name.

get_sourceURL

Returns the URL of the media item. See Media.sourceURL.

getAttributeName

Returns the name of the attribute corresponding to the specified index. See Media.getAttributeName.

getItemInfo

Returns the value of the specified attribute for the media item. See Media.getItemInfo.

getItemInfoByAtom

Returns the value of the attribute with the specified index number. See Media.getItemInfoByAtom.

getMarkerName

Returns the name of the marker at the specified index. See Media.getMarkerName.

getMarkerTime

Returns the time of the marker at the specified index. See Media.getMarkerTime.

isMemberOf

Returns a value indicating whether the specified media item is a member of the specified playlist. See Media.isMemberOf.

isReadOnlyItem

Returns a value indicating whether the attributes of the specified media item can be edited. See Media.isReadOnlyItem.

put_name

Specifies the name of the media item. See Media.name.

setItemInfo

Sets the value of the specified attribute for the media item. See Media.setItemInfo.

Retrieve a pointer to an IWMPMedia interface with the following methods.
The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPMedia2 Interface
- IWMPMedia3 Interface

IWMPMedia2 Interface

The IWMPMedia2 interface provides a method that supplements the IWMPMedia interface.

In addition to the methods inherited from IWMPMedia, the IWMPMedia2 interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_error</td>
<td>Retrieves an IWMPErrorItem pointer if the media item has an error condition. See Media.error.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an IWMPMedia2 interface by calling the QueryInterface method of an IWMPMedia interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
The **IWMPMedia3** interface provides methods that supplement the **IWMPMedia2** interfaces.

In addition to the methods inherited from **IWMPMedia2**, the **IWMPMedia3** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttributeCountByType</td>
<td>Retrieves the number of attributes associated with the specified attribute type. See <a href="#">Media.getAttributeCountByType</a>.</td>
</tr>
<tr>
<td>getItemInfoByType</td>
<td>Retrieves the value of the attribute corresponding to the specified attribute type and index. See <a href="#">Media.getItemInfoByType</a>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPMedia3** interface by calling the **QueryInterface** method of the **IWMPMedia** interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)
- [IWMPMedia Interface](#)
- [IWMPMedia2 Interface](#)
IWMPMediaCollection Interface

The **IWMPMediaCollection** interface provides methods that do something.

In addition to the methods inherited from **IDispatch**, the **IWMPMediaCollection** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Adds a new media item to <strong>Media Library</strong>. See <strong>MediaCollection.add</strong>.</td>
</tr>
<tr>
<td>getAll</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing all media items in <strong>Media Library</strong>. See <strong>MediaCollection.getAll</strong>.</td>
</tr>
<tr>
<td>getAttributeStringCollection</td>
<td>Retrieves a pointer to an <strong>IWMPStringCollection</strong> interface representing the set of all values for a given attribute within a given media type. See <strong>MediaCollection.getAttributeStringCollection</strong>.</td>
</tr>
<tr>
<td>getByAlbum</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing media items from the specified album. See <strong>MediaCollection.getByAlbum</strong>.</td>
</tr>
<tr>
<td>getByAttribute</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing media items with the specified attribute having the specified value. See <strong>MediaCollection.getByAttribute</strong>.</td>
</tr>
<tr>
<td>getByAuthor</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing media items by the specified author. See <strong>MediaCollection.getByAuthor</strong>.</td>
</tr>
<tr>
<td>getByGenre</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing media items with the specified genre. See <strong>MediaCollection.getByGenre</strong>.</td>
</tr>
<tr>
<td>getName</td>
<td>Retrieves a pointer to an <strong>IWMPPlaylist</strong> interface to an object containing media items with the specified name. See <strong>MediaCollection.getName</strong>.</td>
</tr>
<tr>
<td>getMediaAtom</td>
<td>Retrieves the atom associated with the given property name. See <strong>MediaCollection.getMediaAtom</strong>.</td>
</tr>
<tr>
<td>isDeleted</td>
<td>Retrieves a value indicating whether the specified media item is in the deleted items folder. See <strong>MediaCollection.isDeleted</strong>.</td>
</tr>
</tbody>
</table>
remove

Removes the specified media item from the media collection. See MediaCollection.remove.

setDeleted

Moves the specified media item to the deleted items folder. See MediaCollection.setDeleted.

Retrieve a pointer to an IWMPMediaCollection interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCore</td>
<td>get_mediaCollection. See Player.mediaCollection.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces

IWMPMetadataPicture Interface

The IWMPMetadataPicture interface provides methods for retrieving information about the WM/Picture metadata attribute.

In addition to the methods inherited from IDispatch, the IWMPMetadataPicture interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_description</td>
<td>Retrieves a description of the metadata image. See MetadataPicture.description.</td>
</tr>
<tr>
<td>get_mimeType</td>
<td>Retrieves the MIME type of the metadata image. See MetadataPicture.mimeType.</td>
</tr>
<tr>
<td>get_pictureType</td>
<td>Retrieves the picture type of the metadata image. See MetadataPicture.pictureType.</td>
</tr>
<tr>
<td>get_URL</td>
<td>Retrieves the URL of the metadata image. See</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an `IWMPMetadataPicture` interface with the following method.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- [C++ Translation Guide](#)
- [Interfaces](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPMetadataText Interface**

The `IWMPMetadataText` interface provides methods for retrieving information about complex textual metadata attributes.

In addition to the methods inherited from `IDispatch`, the `IWMPMetadataText` interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_description</code></td>
<td>Retrieves a description of the metadata text. See <code>MetadataText.description</code>.</td>
</tr>
<tr>
<td><code>get_text</code></td>
<td>Retrieves the metadata text. See <code>MetadataText.text</code>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an `IWMPMetadataText` interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IWMPMedia3</code></td>
<td><code>getItemInfoByType</code>. See <code>Media.getItemInfoByType</code>.</td>
</tr>
</tbody>
</table>
The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces

IWMPNetwork Interface

The IWMPNetwork interface provides methods relating to the network connection of Windows Media Player.

In addition to the methods inherited from IDispatch, the IWMPNetwork interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_bandWidth</td>
<td>Retrieves the current bandwidth of the clip. See Network.bandWidth.</td>
</tr>
<tr>
<td>get_bitRate</td>
<td>Retrieves the current bit rate being received. See Network.bitRate.</td>
</tr>
<tr>
<td>get_bufferingCount</td>
<td>Retrieves the number of times buffering occurred during clip playback. See Network.bufferingCount.</td>
</tr>
<tr>
<td>get_bufferingProgress</td>
<td>Retrieves the percentage of buffering completed. See Network.bufferingProgress.</td>
</tr>
<tr>
<td>get_bufferingTime</td>
<td>Retrieves the amount of buffering time in milliseconds before playing begins. See Network.bufferingTime.</td>
</tr>
<tr>
<td>get_downloadProgress</td>
<td>Retrieves the percentage of download completed. See Network.downloadProgress.</td>
</tr>
<tr>
<td>get_encodedFrameRate</td>
<td>Retrieves the video frame rate specified by the content author. See Network_encodedFrameRate.</td>
</tr>
<tr>
<td>get_frameRate</td>
<td>Retrieves the current video frame rate. See Network.frameRate.</td>
</tr>
<tr>
<td>get_framesSkipped</td>
<td>Retrieves the total number of frames skipped during playback. See Network.framesSkipped.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an IWMPNetwork interface with the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_lostPackets</td>
<td>Retrieves the number of packets lost. See Network.lostPackets.</td>
</tr>
<tr>
<td>get_maxBandwidth</td>
<td>Retrieves the maximum allowed bandwidth. See Network.maxBandwidth.</td>
</tr>
<tr>
<td>get_maxBitRate</td>
<td>Retrieves the maximum possible video bit rate. See Network.maxBitRate.</td>
</tr>
<tr>
<td>get_receivedPackets</td>
<td>Retrieves the number of packets received. See Network.receivedPackets.</td>
</tr>
<tr>
<td>get_receptionQuality</td>
<td>Retrieves the percentage of packets received in the last 30 seconds. See Network.receptionQuality.</td>
</tr>
<tr>
<td>get_recoveredPackets</td>
<td>Retrieves the number of recovered packets. See Network.recoveredPackets.</td>
</tr>
<tr>
<td>get_sourceProtocol</td>
<td>Retrieves the source protocol used to receive the data. See Network.sourceProtocol.</td>
</tr>
<tr>
<td>getProxyBypassForLocal</td>
<td>Retrieves a value indicating whether the proxy server should be bypassed if the origin server is on a local network. See Network.getProxyBypassForLocal.</td>
</tr>
<tr>
<td>getProxyExceptionList</td>
<td>Retrieves the proxy exception list. See Network.getProxyExceptionList.</td>
</tr>
<tr>
<td>getProxyName</td>
<td>Retrieves the name of a proxy server to use. See Network.getProxyName.</td>
</tr>
<tr>
<td>getProxyPort</td>
<td>Retrieves the proxy port to use. See Network.getProxyPort.</td>
</tr>
<tr>
<td>getProxySettings</td>
<td>Retrieves the proxy settings for a given protocol. See Network.getProxySettings.</td>
</tr>
<tr>
<td>put_bufferingTime</td>
<td>Specifies the amount of buffering time in milliseconds before playing begins. See Network.bufferingTime.</td>
</tr>
<tr>
<td>put_maxBandwidth</td>
<td>Specifies the maximum allowed bandwidth. See Network.maxBandwidth.</td>
</tr>
<tr>
<td>setProxyExceptionList</td>
<td>Specifies the proxy exception list. See Network.setProxyExceptionList.</td>
</tr>
<tr>
<td>setProxyName</td>
<td>Specifies the name of a proxy server to use. See Network.setProxyName.</td>
</tr>
<tr>
<td>setProxyPort</td>
<td>Specifies the proxy port to use. See Network.setProxyPort.</td>
</tr>
<tr>
<td>setProxySettings</td>
<td>Specifies the proxy settings for a given protocol. See Network.setProxySettings.</td>
</tr>
</tbody>
</table>
The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also
- C++ Translation Guide
- Interfaces

IWMPPlayer Interface

The **IWMPPlayer** interface provides methods for modifying the basic behavior of the control user interface. These methods supplement the **IWMPCore** interface.

In addition to the methods inherited from **IWMPCore**, the **IWMPPlayer** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_enableContextMenu</td>
<td>Retrieves a value indicating whether to enable the context menu, which appears when the right mouse button is clicked. See <strong>Player.enableContextMenu</strong>.</td>
</tr>
<tr>
<td>get_enabled</td>
<td>Retrieves a value indicating whether the Windows Media Player control is enabled. See <strong>Player.enabled</strong>.</td>
</tr>
<tr>
<td>get_fullScreen</td>
<td>Retrieves a value indicating whether video content is played back in full-screen mode. See <strong>Player.fullScreen</strong>.</td>
</tr>
<tr>
<td>get_uiMode</td>
<td>Retrieves a value indicating which controls are shown in the user interface when Windows Media Player is embedded in a Web page. See <strong>Player.uiMode</strong>.</td>
</tr>
<tr>
<td>put_enableContextMenu</td>
<td>Specifies a value indicating whether to enable the context menu, which appears when the right mouse button is clicked. See <strong>Player.enableContextMenu</strong>.</td>
</tr>
<tr>
<td>put_enabled</td>
<td>Specifies a value indicating whether the Windows...</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an IWMPPlayer interface by calling the COM CoCreateInstance method.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for IWMPPlayer4, which inherits from this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPCore Interface
- IWMPCore2 Interface
- IWMPCore3 Interface
- IWMPPlayer2 Interface
- IWMPPlayer3 Interface
- IWMPPlayer4 Interface

### IWMPPlayer2 Interface

The IWMPPlayer2 interface provides additional methods for modifying the basic behavior of the control user interface. These methods also supplement the IWMPCore interface.

The IWMPPlayer2 interface duplicates the methods of IWMPPlayer, inherits the methods of IWMPCore, and exposes the following additional methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_stretchToFit</td>
<td>Retrieves a value indicating whether video will stretch to fit size of the Windows Media Player control video</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an **IWMPPlayer2** interface either by calling the **QueryInterface** method the **IWMPPlayer** interface or by calling the COM **CoCreateInstance** method.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for **IWMPPlayer4**, which inherits from this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)
- [IWMPCore Interface](#)
- [IWMPCore2 Interface](#)
- [IWMPCore3 Interface](#)
- [IWMPPlayer Interface](#)
- [IWMPPlayer3 Interface](#)
- [IWMPPlayer4 Interface](#)

---

**IWMPPlayer3 Interface**

The **IWMPPlayer3** interface provides methods for modifying the basic behavior of the control user interface. These methods supplement the **IWMPCore2** interface.

The **IWMPPlayer3** interface duplicates the methods of **IWMPPlayer** and **IWMPPlayer2** and inherits the methods of **IWMPCore2**. It is identical to **IWMPPlayer2** except for the inherited interface.
Retrieve a pointer to an **IWMPPlayer3** interface either by calling the **QueryInterface** method the **IWMPPlayer** or **IWMPPlayer2** interfaces or by calling the COM **CoCreateInstance** method.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface. They do automatically generate a wrapper class for **IWMPPlayer4**, which inherits from this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)
- [IWMPCore Interface](#)
- [IWMPCore2 Interface](#)
- [IWMPCore3 Interface](#)
- [IWMPPlayer Interface](#)
- [IWMPPlayer2 Interface](#)
- [IWMPPlayer4 Interface](#)

## IWMPPlayer4 Interface

The **IWMPPlayer4** interface provides methods for modifying the basic behavior of the control user interface. These methods supplement the **IWMPCore3** interface.

The **IWMPPlayer4** interface duplicates the methods of **IWMPPlayer**, **IWMPPlayer2**, and **IWMPPlayer3**, inherits the methods of **IWMPCore3**, and exposes the following additional methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_isRemote</td>
<td>Retrieves a value indicating whether the Windows Media Player control is running in remote mode. See <strong>Player.isRemote</strong>.</td>
</tr>
<tr>
<td>get_playerApplication</td>
<td>Retrieves an <strong>IWMPPlayerApplication</strong> pointer when a remoted Windows Media Player control is running. See <strong>Player.playerApplication</strong>.</td>
</tr>
<tr>
<td>openPlayer</td>
<td>Opens Windows Media Player using the specified URL. See <strong>Player.openPlayer</strong>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPPlayer4** interface either by calling the **QueryInterface** method of an
IWMPPlayer interface or by calling the COM CoCreateInstance method.

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
- IWMPCore Interface
- IWMPCore2 Interface
- IWMPCore3 Interface
- IWMPPlayer Interface
- IWMPPlayer2 Interface
- IWMPPlayer3 Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPPlayerApplication Interface

The IWMPPlayerApplication interface provides methods for switching between a remoted Windows Media Player control and the full mode of the Player. These methods can only be used with C++ programs that embed the control in remote mode.

In addition to the methods inherited from IDispatch, the IWMPPlayerApplication interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_hasDisplay</td>
<td>Retrieves a value indicating whether video can display through the remoted Player control. See PlayerApplication.hasDisplay.</td>
</tr>
<tr>
<td>get_playerDocked</td>
<td>Retrieves a value indicating whether the Player is in a docked state. See PlayerApplication.playerDocked.</td>
</tr>
<tr>
<td>switchToControl</td>
<td>Switches a remoted Player control to the docked state. See PlayerApplication.switchToControl.</td>
</tr>
<tr>
<td>switchToPlayerApplication</td>
<td>Switches a remoted Player control to the full mode of the Player. See PlayerApplication.switchToPlayerApplication.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an **IWMPPlayerApplication** interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPPlayer4</td>
<td>get_playerApplication. See</td>
</tr>
<tr>
<td></td>
<td>Player.playerApplication.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

**See Also**

- C++ Translation Guide
- Interfaces

---

**IWMPPlayerServices Interface**

The **IWMPPlayerServices** interface provides methods used by the host of a remoted control to manipulate the full mode of the Player. These methods can only be used with C++.

In addition to the methods inherited from **IUnknown**, the **IWMPPlayerServices** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateUIPlugin</td>
<td>Activates the specified user interface plug-in in the full mode of the Player.</td>
</tr>
<tr>
<td>setTaskPane</td>
<td>Displays the specified task pane in the full mode of the Player.</td>
</tr>
<tr>
<td>setTaskPaneURL</td>
<td>Displays the specified URL in the specified task pane of the full mode of the Player.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPPlayerServices** interface by calling the COM **CoCreateInstance** method.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

**See Also**
IWMPPlayerServices::activateUIPlugin

The activateUIPlugin method activates the specified UI plug-in in the full mode of the Player.

Syntax

HRESULT activateUIPlugin(
    BSTR bstrPlugin
);

Parameters

bstrPlugin

[in] BSTR containing the name of the plug-in to activate.

Return Values

The method returns an HRESULT. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

Remarks

This method is used only when remoting the Player control.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also
IWMPlayerServices::setTaskPane

The `setTaskPane` method displays the specified task pane in the full mode of the Player.

Syntax

```c
HRESULT setTaskPane(
    BSTR bstrTaskPane
);
```

Parameters

`bstrTaskPane`

[in] BSTR containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NowPlaying</td>
<td>Opens the Player in the Now Playing feature.</td>
</tr>
<tr>
<td>MediaGuide</td>
<td>Opens the Player in the Media Guide feature.</td>
</tr>
<tr>
<td>CopyFromCD</td>
<td>Opens the Player in the Copy From CD feature.</td>
</tr>
<tr>
<td>CopyFromCD?AutoCopy:id</td>
<td>Opens the Player in the Copy From CD feature and automatically invokes the Copy Music functionality after switching. To specify a particular drive identifier, append a colon character (:) followed by the CD-ROM drive identifier number. If you omit the colon and the id, the first CD-ROM drive is will be used. If the user has selected Eject CD when copying is completed in Windows Media Player, the CD will be ejected when copying is completed.</td>
</tr>
<tr>
<td>MediaLibrary</td>
<td>Opens the Player in the Media Library feature.</td>
</tr>
</tbody>
</table>
RadioTuner
Opens the Player in the **Radio Tuner** feature.

CopyToCDOrDevice
Opens the Player in the **Copy to CD or Device** feature.

Services
Opens the Player in the **Premium Services** feature.

SkinChooser
Opens the Player in the **Skin Chooser** feature.

**Return Values**

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

**Remarks**

This method is used only when remoting the Player control.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmp.idl; include wmp.h.

**Library**: Use wmp.dll.

**See Also**

- **IWMPPPlayerServices Interface**
- **Remoting the Windows Media Player Control**
- **setTaskPaneURL**

**IWMPPlayerServices::setTaskPaneURL**

The **setTaskPaneURL** method displays the specified URL in the specified task pane of the full mode of the Player.
Syntax

```c
HRESULT setTaskPaneURL(
    BSTR bstrTaskPane,
    BSTR bstrURL,
    BSTR bstrFriendlyName
);
```

Parameters

*bstrTaskPane*

[in] BSTR containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaGuide</td>
<td>Opens the Player in the <strong>MediaGuide</strong> feature.</td>
</tr>
<tr>
<td>RadioTuner</td>
<td>Opens the Player in the <strong>RadioTuner</strong> feature.</td>
</tr>
<tr>
<td>Services</td>
<td>Opens the Player in the <strong>Premium Services</strong> feature.</td>
</tr>
</tbody>
</table>

*bstrURL*

[in] BSTR containing the URL to display in the task pane.

*bstrFriendlyName*

[in] BSTR containing the friendly name of the content at the specified URL.

Return Values

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

Remarks

This method is used only when remoting the Player control. This method must be called when the control is in the docked state. Once set, the specified task pane is opened the next time the remoted control transitions to the Player application.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

**Library:** Use wmp.dll.

See Also
IWMPPlaylist Interface

The IWMPPlaylist interface provides methods for manipulating lists of media clips.

In addition to the methods inherited from IDispatch, the IWMPPlaylist interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendItem</td>
<td>Adds a media item to the end of the playlist. See Playlist.appendItem.</td>
</tr>
<tr>
<td>clear</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>get_attributeCount</td>
<td>Retrieves the number of attributes associated with the playlist. See Playlist.attributeCount.</td>
</tr>
<tr>
<td>get_attributeName</td>
<td>Retrieves the name of an attribute specified by an index. See Playlist.attributeName.</td>
</tr>
<tr>
<td>get_count</td>
<td>Retrieves the number of items in the playlist. See Playlist.count.</td>
</tr>
<tr>
<td>get_name</td>
<td>Retrieves the name of the playlist. See Playlist.name.</td>
</tr>
<tr>
<td>getItemInfo</td>
<td>Retrieves the value of a playlist attribute specified by name. See Playlist.getItemInfo.</td>
</tr>
<tr>
<td>insertItem</td>
<td>Adds a media item at the specified location in the playlist. See Playlist.insertItem.</td>
</tr>
<tr>
<td>isIdentical</td>
<td>Retrieves a value indicating whether the specified playlist is identical to the current playlist. See Playlist.isIdentical.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves the media item at the specified index. See Playlist.item.</td>
</tr>
<tr>
<td>moveItem</td>
<td>Changes the location of a media item in the playlist.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an **IWMPPlaylist** interface with the following methods.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCdrom</td>
<td>get_playlist. See <strong>Cdrom.playlist</strong></td>
</tr>
<tr>
<td>IWMPCore</td>
<td>get_currentPlaylist. See <strong>Player.currentPlaylist</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getAll. See <strong>MediaCollection.getAll</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getByAlbum. See <strong>MediaCollection.getByAlbum</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getByAttribute. See <strong>MediaCollection.getByAttribute</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getByAuthor. See <strong>MediaCollection.getByAuthor</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getByGenre. See <strong>MediaCollection.getByGenre</strong></td>
</tr>
<tr>
<td>IWMPMediaCollection</td>
<td>getName. See <strong>MediaCollection.getByName</strong></td>
</tr>
<tr>
<td>IWMPPlaylistArray</td>
<td>item. See <strong>PlaylistArray.item</strong></td>
</tr>
<tr>
<td>IWMPPlaylistCollection</td>
<td>newPlaylist. See <strong>PlaylistCollection.newPlaylist</strong></td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

**See Also**

- [C++ Translation Guide](#)
- [Interfaces](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
The **IWMPPlaylistArray** interface provides methods for accessing a collection of **IWMPPlaylist** pointers by index number.

In addition to the methods inherited from **IDispatch**, the **IWMPPlaylistArray** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get_count</strong></td>
<td>Retrieves the number of playlists in the playlist array. See <strong>PlaylistArray.count</strong>.</td>
</tr>
<tr>
<td><strong>item</strong></td>
<td>Retrieves a pointer to the <strong>IWMPPlaylist</strong> interface representing the playlist at the specified index. See <strong>PlaylistArray.item</strong>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPPlaylistArray** interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IWMPPPlaylistCollection</strong></td>
<td><strong>getAll</strong>. See <strong>PlaylistCollection.getAll</strong>.</td>
</tr>
<tr>
<td><strong>IWMPPPlaylistCollection</strong></td>
<td><strong>getByName</strong>. See <strong>PlaylistCollection.getByName</strong>.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- [C++ Translation Guide](#)
- [Interfaces](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPPPlaylistCollection Interface**

The **IWMPPPlaylistCollection** interface provides methods for manipulating **IWMPPlaylist** and **IWMPPlaylistArray** pointers.

In addition to the methods inherited from **IDispatch**, the **IWMPPPlaylistCollection** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
Retrieve a pointer to an IWMPPlaylistCollection interface with the following method.

### Interface Method

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPCore</td>
<td>get_playlistCollection</td>
<td>Retrieves a pointer to an IWMPPlaylistCollection interface.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
- Managing Playlists
The **IWMPRemoteMediaServices** interface includes methods that provide services to the Player from a program that hosts the Player control. These methods can only be used with C++, and some can only be used with remoting.

In addition to the methods inherited from **IUnknown**, the **IWMPRemoteMediaServices** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetApplicationName</td>
<td>Called by Windows Media Player to retrieve the name of the program that is hosting the remoted control.</td>
</tr>
<tr>
<td>GetCustomUIMode</td>
<td>Called by Windows Media Player to retrieve the location of a skin file to apply to the Player control.</td>
</tr>
<tr>
<td>GetScriptableObject</td>
<td>Called by Windows Media Player to retrieve a name and interface pointer for an object that can be called from the script code within a skin.</td>
</tr>
<tr>
<td>GetServiceType</td>
<td>Called by Windows Media Player to determine whether a host program wants to run its embedded control remotely.</td>
</tr>
</tbody>
</table>

A pointer to an **IWMPRemoteMediaServices** interface is retrieved by calling the COM **CoCreateInstance** method.

The MFC AppWizards in Visual Studio do not automatically generate a separate wrapper class for this interface.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmp.idl; include wmp.h.

**See Also**

- Interfaces
- Remoting the Windows Media Player Control

---

© 2000-2003 Microsoft Corporation. All rights reserved.
The **GetApplicationName** method is called by Windows Media Player to retrieve the name of the program that is hosting the remoted control.

### Syntax

```c
HRESULT GetApplicationName(
    BSTR* pbstrName
);
```

### Parameters

`pbstrName`

[out] Pointer to a **BSTR** containing the name of the host program.

### Return Values

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

### Remarks

This method is used only when remoting the Player control.

The full mode of the Player displays the program name on the **View** menu under **Switch to Other Program**.

### Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmp.idl; include wmp.h.

### See Also

- [IWMPRemoteMediaServices Interface](#)
- [Remoting the Windows Media Player Control](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
The GetCustomUIMode method is called by Windows Media Player to retrieve the location of a skin file to apply to the Player control.

**Syntax**

```c
HRESULT GetCustomUIMode(
    BSTR* pbstrFile
);
```

**Parameters**

- `pbstrFile` 
  [out] Pointer to a `BSTR` containing the location of the skin file.

**Return Values**

The method returns an `HRESULT`. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

**Remarks**

This method allows you to customize the user interface of the embedded control using Windows Media Player skin technology. Skins used in this way can communicate with the host of the control through a scriptable object retrieved automatically by Windows Media Player using the `IWMPRemoteMediaServices::GetScriptableObject` method.

To apply a skin file to the embedded control, you must call `IWMPPlayer.put_uiMode` with a value of "custom". Your implementation of `GetCustomUIMode` must also return a value of "file://" followed by the path and file name of a skin definition file located on the local computer.

The embedded Player control does not have to be remoted to use this method.

**Requirements**

- **Version:** Windows Media Player 9 Series or later.
- **Header:** Defined in wmp.idl; include wmp.h.

**See Also**

- [IWMPRemoteMediaServices Interface](#)
- [IWMPRemoteMediaServices::GetScriptableObject](#)
- [Using Skins with the Windows Media Player Control](#)
IWMPRemoteMediaServices::GetScriptableObject

The `GetScriptableObject` method is called by Windows Media Player to retrieve a name and interface pointer for an object that can be called from the script code within a skin.

**Syntax**

```c
HRESULT GetScriptableObject(
    BSTR* pbstrName,
    IDispatch** ppDispatch
);
```

**Parameters**

- `pbstrName`  
  [out] Pointer to a `BSTR` containing the name of the scriptable object.

- `ppDispatch`  
  [out] Pointer to a pointer to the `IDispatch` interface of the scriptable object.

**Return Values**

The method returns an `HRESULT`. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

**Remarks**

The user interface of an embedded control can be customized by using Windows Media Player skin technology. You must specify a skin file location by using the `IWMPRemoteMediaServices::GetCustomUIMode` method. Skins used in this way can communicate with the host of the control through a scriptable object retrieved automatically by Windows Media Player by using the `IWMPRemoteMediaServices::GetScriptableObject` method.

The embedded Player control does not have to be remoted to use this method.

**Requirements**

**Version:** Windows Media Player 9 Series or later.
**GetServiceType**

The **GetServiceType** method is called by Windows Media Player to determine whether a host program wants to run its embedded control remotely.

**Syntax**

```cpp
HRESULT GetServiceType(
    BSTR* pbstrType
);
```

**Parameters**

*pbstrType*

[out] Pointer to a **BSTR** containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>The Player control is embedded in local mode.</td>
</tr>
<tr>
<td>Remote</td>
<td>The Player control is embedded in remote mode.</td>
</tr>
<tr>
<td>RemoteNoDialogs</td>
<td>The Player control is embedded in remote mode and does not display dialog boxes. Use of this value requires Windows Media Player 9 Series update 819756 or later.</td>
</tr>
</tbody>
</table>

**Return Values**

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
The method succeeded.

Requirements

Version: Windows Media Player 9 Series or later. Update 819756 or later when using RemoteNoDialogs.

Header: Defined in wmp.idl; include wmp.h.

See Also

- IWMPRemoteMediaServices Interface
- Remoting the Windows Media Player Control

IWMPSettings Interface

The IWMPSettings interface provides a way to modify various settings of Windows Media Player.

In addition to the methods inherited from IDispatch, the IWMPSettings interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_autoStart</td>
<td>Retrieves a value indicating whether the current media item begins playing automatically. See Settings.autoStart.</td>
</tr>
<tr>
<td>get_balance</td>
<td>Retrieves the current stereo balance. See Settings.balance.</td>
</tr>
<tr>
<td>get_baseURL</td>
<td>Retrieves the base URL used for relative path resolution with URL-type script commands embedded in media files. See Settings.baseURL.</td>
</tr>
<tr>
<td>get_defaultFrame</td>
<td>Retrieves the name of the frame used to display a URL received in a scriptCommand event. See Settings.defaultFrame.</td>
</tr>
<tr>
<td>get_enableErrorDialogs</td>
<td>Retrieves a value indicating whether error dialog boxes are shown automatically. See Settings.enableErrorDialogs.</td>
</tr>
</tbody>
</table>
Retrieve a pointer to an `IWMPSettings` interface with the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_invokeURLs</code></td>
<td>Retrieves a value indicating whether URL events should launch a Web browser. See <code>Settings.invokeURLs</code>.</td>
</tr>
<tr>
<td><code>get_isAvailable</code></td>
<td>Retrieves a value indicating whether a specified type of information is available or a given action can be performed. See <code>Settings.isAvailable</code>.</td>
</tr>
<tr>
<td><code>get_mute</code></td>
<td>Retrieves a value indicating whether audio is muted. See <code>Settings.mute</code>.</td>
</tr>
<tr>
<td><code>get_playCount</code></td>
<td>Retrieves the number of times a media item will play. See <code>Settings.playCount</code>.</td>
</tr>
<tr>
<td><code>get_rate</code></td>
<td>Retrieves the current playback rate. See <code>Settings.rate</code>.</td>
</tr>
<tr>
<td><code>get_volume</code></td>
<td>Retrieves the current volume. See <code>Settings.volume</code>.</td>
</tr>
<tr>
<td><code>getMode</code></td>
<td>Returns a value indicating whether loop mode or shuffle mode is active. See <code>Settings.getMode</code>.</td>
</tr>
<tr>
<td><code>put_autoStart</code></td>
<td>Specifies a value indicating whether the current media item begins playing automatically. See <code>Settings.autoStart</code>.</td>
</tr>
<tr>
<td><code>put_balance</code></td>
<td>Specifies the current stereo balance. See <code>Settings.balance</code>.</td>
</tr>
<tr>
<td><code>put_baseURL</code></td>
<td>Specifies the base URL used for relative path resolution with URL-type script commands embedded in media files. See <code>Settings.baseURL</code>.</td>
</tr>
<tr>
<td><code>put_defaultFrame</code></td>
<td>Specifies the name of the frame used to display a URL received in a <code>scriptCommand</code> event. See <code>Settings.defaultFrame</code>.</td>
</tr>
<tr>
<td><code>put_enableErrorDialogs</code></td>
<td>Specifies a value indicating whether error dialog boxes are shown automatically. See <code>Settings.enableErrorDialogs</code>.</td>
</tr>
<tr>
<td><code>put_invokeURLs</code></td>
<td>Specifies a value indicating whether URL events should launch a Web browser. See <code>Settings.invokeURLs</code>.</td>
</tr>
<tr>
<td><code>put_mute</code></td>
<td>Specifies a value indicating whether audio is muted. See <code>Settings.mute</code>.</td>
</tr>
<tr>
<td><code>put_playCount</code></td>
<td>Specifies the number of times a media item will play. See <code>Settings.playCount</code>.</td>
</tr>
<tr>
<td><code>put_rate</code></td>
<td>Specifies the current playback rate. See <code>Settings.rate</code>.</td>
</tr>
<tr>
<td><code>put_volume</code></td>
<td>Specifies the current volume. See <code>Settings.volume</code>.</td>
</tr>
<tr>
<td><code>setMode</code></td>
<td>Sets the loop mode or shuffle mode to active or inactive. See <code>Settings.setMode</code>.</td>
</tr>
</tbody>
</table>
The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces

### IWMPSettings2 Interface

The **IWMPSettings2** interface provides methods that supplement the **IWMPSettings** interface.

In addition to the methods inherited from **IWMPSettings**, the **IWMPSettings2** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get_defaultAudioLanguage</strong></td>
<td>Retrieves the locale identifier (LCID) of the default audio language. See <a href="#">Settings.defaultAudioLanguage</a>.</td>
</tr>
<tr>
<td><strong>get_mediaAccessRights</strong></td>
<td>Retrieves a value indicating the rights currently granted for Media Library access. See <a href="#">Settings.mediaAccessRights</a>.</td>
</tr>
<tr>
<td><strong>requestMediaAccessRights</strong></td>
<td>Requests a specified level of access to Media Library. See <a href="#">Settings.requestMediaAccessRights</a>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPSettings2** interface by calling the **QueryInterface** method of the **IWMPSettings** interface.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces
IWMPSkinManager Interface

The IWMPSkinManager interface provides a method used to synchronize the current skin with the current desktop theme in Microsoft Windows XP. This method can only be used with C++.

In addition to the methods inherited from IDispatch, the IWMPSkinManager interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetVisualStyle</td>
<td>Specifies the path to a theme file in Windows XP to which Windows Media Player synchronizes the skin.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an IWMPSkinManager interface by calling the COM CoCreateInstance method.

The MFC AppWizards in Visual Studio do not automatically generate a wrapper class for this interface.

See Also

- Interfaces

IWMPSkinManager::SetVisualStyle

The SetVisualStyle method specifies the path to a theme file in Windows XP to which Windows Media Player synchronizes the skin.
Syntax

```c
HRESULT SetVisualStyle(
    BSTR bstrStyle
);
```

Parameters

*bstrStyle*

[in] BSTR containing the path to the theme file.

Return Values

The method returns an HRESULT. Possible values include, but are not limited to, those in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>The method succeeded.</td>
</tr>
</tbody>
</table>

Remarks

Windows XP calls this method when the user changes the current theme. The current skin selection will change to match the theme, or will change to the Windows Classic skin if there is no skin that matches the current theme. If the Player is in skin mode, the skin will change immediately. Otherwise, the new skin selection will be applied the next time skin mode is entered.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- [IWMPSkinManager Interface](#)
IWMPStringCollection Interface

The **IWMPStringCollection** interface provides methods that work with a collection of strings.

In addition to the methods inherited from **IDispatch**, the **IWMPStringCollection** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_count</td>
<td>Retrieves the number of items in the string collection. See <strong>StringCollection.count</strong>.</td>
</tr>
<tr>
<td>item</td>
<td>Retrieves the string at the specified index. See <strong>StringCollection.item</strong>.</td>
</tr>
</tbody>
</table>

Retrieve a pointer to an **IWMPStringCollection** interface with the following method.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPMediaCollection</td>
<td>getAttributeStringCollection. See <strong>MediaCollection.getAttributeStringCollection</strong>.</td>
</tr>
</tbody>
</table>

The MFC AppWizards in Visual Studio automatically generate a wrapper class for this interface.

See Also

- C++ Translation Guide
- Interfaces

Enumeration Types

The following enumeration types are available in C++ code.

<table>
<thead>
<tr>
<th>Enumeration type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMPOpenState</td>
<td>Defines the possible operational states of Windows Media Player as it opens a digital media file.</td>
</tr>
</tbody>
</table>
The **WMPOpenState** enumeration type defines the possible operational states of Windows Media Player as it opens a digital media file.

**Syntax**

```c
typedef enum WMPOpenState {  
    wmposUndefined             = 0,  
    wmposPlaylistChanging      = 1,  
    wmposPlaylistLocating      = 2,  
    wmposPlaylistConnecting    = 3,  
    wmposPlaylistLoading       = 4,  
    wmposPlaylistOpening       = 5,  
    wmposPlaylistOpenNoMedia   = 6,  
    wmposPlaylistChanged       = 7,  
    wmposMediaChanging         = 8,  
    wmposMediaLocating         = 9,  
    wmposMediaConnecting       = 10,  
    wmposMediaLoading          = 11,  
    wmposMediaOpening          = 12,  
    wmposMediaOpen             = 13,  
    wmposBeginCodecAcquisition = 14,  
    wmposEndCodecAcquisition   = 15,  
    wmposBeginLicenseAcquisition = 16,  
    wmposEndLicenseAcquisition = 17,  
    wmposBeginIndividualization = 18,  
    wmposEndIndividualization  = 19,  
    wmposMediaWaiting          = 20,  
    wmposOpeningUnknownURL     = 21  
} WMPOpenState;
```

**Members**
wmposUndefined
The content source is in an undefined state.

wmposPlaylistChanging
A new playlist is about to be loaded.

wmposPlaylistLocating
Locating the playlist.

wmposPlaylistConnecting
Connecting to the server that is hosting the playlist.

wmposPlaylistLoading
Loading a playlist.

wmposPlaylistOpening
Opening a playlist.

wmposPlaylistOpenNoMedia
Playlist is open.

wmposPlaylistChanged
Playlist has changed.

wmposMediaChanging
New media is about to be loaded.

wmposMediaLocating
Locating the media.

wmposMediaConnecting
Connecting to the server that is hosting the media.

wmposMediaLoading
Loading the media.

wmposMediaOpening
Opening the media.
wmposMediaOpen

Media is open.

wmposBeginCodecAcquisition

Starting codec acquisition.

wmposEndCodecAcquisition

Ending codec acquisition.

wmposBeginLicenseAcquisition

Starting license acquisition.

wmposEndLicenseAcquisition

Ending license acquisition.

wmposBeginIndividualization

Starting individualization.

wmposEndIndividualization

Individualization has ended.

wmposMediaWaiting

Waiting for media.

wmposOpeningUnknownURL

Opening an URL whose type is unknown.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Enumeration Types
WMPPlaylistChangeEventType

The `WMPPlaylistChangeEventType` enumeration type defines the types of changes that can be made to a playlist.

**Syntax**

```c
typedef enum WMPPlaylistChangeEventType{
    wmplcUnknown    = 0,
    wmplcClear      = 1,
    wmplcInfoChange = 2,
    wmplcMove       = 3,
    wmplcDelete     = 4,
    wmplcInsert     = 5,
    wmplcAppend     = 6,
    wmplcPrivate    = 7,
    wmplcNameChange = 8,
    wmplcMorph      = 9,
    wmplcSort       = 10
} WMPPlaylistChangeEventType;
```

**Members**

- **wmplcUnknown**
  An unknown change has occurred.

- **wmplcClear**
  The playlist has been cleared.

- **wmplcInfoChange**
  A playlist attribute has changed value.

- **wmplcMove**
  A media item within the playlist has been moved to a new position.

- **wmplcDelete**
  A media item has been deleted from the playlist.

- **wmplcInsert**
  A media item has been inserted into the playlist.
wmpcAppend

A media item has been appended to the playlist.

wmpcPrivate

Not supported.

wmpcNameChange

The name of the playlist has changed.

wmpcMorph

Not supported.

wmpcSort

The playlist has been sorted.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Enumeration Types

WMPPlayState

The WMPPlayState enumeration type defines the possible operational states of Windows Media Player as it plays a digital media file.

Syntax
typedef enum WMPPlayState{
    wmppsUndefined     = 0,
    wmppsStopped       = 1,
    wmppsPaused        = 2,
    wmppsPlaying       = 3,
    wmppsScanForward   = 4,
    wmppsScanReverse   = 5,
    wmppsBuffering     = 6,
    wmppsWaiting       = 7,
    wmppsMediaEnded    = 8,
    wmppsTransitioning = 9,
    wmppsReady         = 10,
    wmppsReconnecting  = 11
} WMPPlayState;

Members

wmppsUndefined

Windows Media Player is in an undefined state.

wmppsStopped

Playback is stopped.

wmppsPaused

Playback is paused.

wmppsPlaying

Stream is playing.

wmppsScanForward

Stream is scanning forward.

wmppsScanReverse

Stream is scanning in reverse.

wmppsBuffering

Media is being buffered.

wmppsWaiting

Waiting for streaming data.

wmppsMediaEnded

The end of the media has been reached.

wmppsTransitioning
Preparing new media.

wmppsReady

Ready to begin playing.

wmppsReconnecting

Trying to reconnect for streaming data.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in wmp.idl; include wmp.h.

Library: Use wmp.dll.

See Also

- Enumeration Types

Windows Media Player Skins

Microsoft Windows Media Player provides a programming platform to create custom skins. Skins are sets of scripts, art, media, and text files that can be combined to create a new appearance for Windows Media Player. Using skins, you can change not only the way Windows Media Player looks, but how it functions. Not just where the knobs are and what they look like, but what they do, given the limits of the underlying Windows Media Player core technology.

Skin technology is very different from other kinds of programming: essentially you will be mixing programming and art in equal parts. You do not need to be an expert programmer (not much more than you already know if you have created Web pages and done some simple scripting), and neither do you need to be an artist (as long as you can use an art manipulation program such as Adobe Photoshop). You'll be using XML (similar to HTML), Microsoft JScript (similar to Microsoft Visual Basic and other high-level languages), and whatever art programs you choose.

The skins documentation contains the following three sections.
About Skins

Microsoft Windows Media Player lets the user choose from a variety of standard skins, each one providing an additional visual experience that enhances listening and viewing pleasure. Windows Media Player comes with several skins to choose from, but it is relatively easy to create and distribute custom skins.

Skins are simply collections of one or more files of computer art, organized by a text file, that tells Windows Media Player how to use these files to display a skin. You can use a variety of programs to modify existing art or create simple art of your own. For example, using the drawing tools and clip art supplied with Microsoft PowerPoint, you can create artwork by just dragging and dropping. Similarly, you do not need to be a programmer to create the text file used to define an individual skin.

One of the reasons that skins are relatively easy to create is that you can look at the individual pieces of art and the source code for any skin, and see how each skin is put together. Several simple example skins will be provided with this SDK, as well as a reference for skin elements and attributes.

The following sections provide overview information about Windows Media Player skins.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why Make Skins?</td>
<td>Describes the some of the uses for skin technology.</td>
</tr>
</tbody>
</table>
Why Make Skins?

When someone uses a program in a graphical computer environment such as Microsoft Windows, the visual options open to them are called a user interface. One of the purposes of Microsoft Windows is to provide a standard user interface so that all programs will operate the same way. For example, Microsoft recommends, in the *Windows User Interface Guidelines*, that every program provide a Close button in the upper right-hand corner of the main window of the program.

Windows Media Player provides the capabilities for creating your own user interface. If you want to put the Close button in the middle of the screen, you can do that. Perhaps you do not like the way the Close button looks (it looks like an X inside a box); if you want it to look like a skull and crossbones, you can make a user interface where the Close button is just that! Windows Media Player gives you all the tools you need to make a custom user interface for playing music and video: buttons, slider bars, video windows, visualization windows, equalization bars, and so on.

There are several good reasons why you might want to create your own user interface for Windows Media Player. One reason is that you might want to add functionality that is not already in Windows Media Player. For example, you might want to create a player that plays music from playlists that are based on the time of day, so that you have upbeat rock in the morning and slow jazz in the evening. Or perhaps you want to make a skin
with a big red button that will stop the music quickly. Windows Media Player does not come with a "play the same song over and over again until my roommate goes crazy" button, but if you want one, you can create it.

Another reason for creating a skin is to make a branded look for Windows Media Player. If you are distributing music from your Web site and use a particular logo, you might want to design a skin that uses your logo to remind people about your site. If you have a rock band, you can make a skin with pictures of your band on it.

And another reason to make skins is to make something unique that can dress up your desktop. When your friends come over and ask you what that cool program on your screen is, you can say you made it yourself. You can even take a picture of your dog, scan it into your computer, add some buttons, and click on your dog's nose to start music and the tail to stop it. You can create different skins for different kinds of music or have a different skin for every day of the week.

See Also

- About Skins

Skin Files

When you see a skin, you are seeing only the surface. Behind the surface are several files that make up the complete skin. Technically speaking, a skin is a group of files, with each file containing a specific kind of information. Here are the types of files you can use to make up a complete skin:

<table>
<thead>
<tr>
<th>File type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skin Definition File</td>
<td>The master file that defines how the other files will be used. This is a text file and has a .wms file name extension.</td>
</tr>
<tr>
<td>Art Files</td>
<td>Art files that contain the graphic elements of your skin. These include BMP, GIF, JPEG, and PNG files.</td>
</tr>
<tr>
<td>JScript Files</td>
<td>If you want to create more complicated responses to events, you can create script files using Microsoft JScript. These are text files and have a .js file name extension.</td>
</tr>
</tbody>
</table>

See Also

- About Skins
Skin Definition File

Skin definition files contain the basic instructions for what the skin does and where other files used by the skin can be found. There can only be one skin definition file for a skin, and it has a .wms file name extension.

The instructions in the skin definition file are written in eXtensible Markup Language (XML), which is similar to HTML. If you have used HTML to create Web pages, you will find that XML looks familiar.

The XML in the skin definition file uses a set of special element tags to define parts of the skin user interface. For example, the BUTTON element defines how a button will behave, where it will go, and what it will look like.

Each element tag has specific attributes. For example, the BUTTON element has an Image attribute that defines where the picture of the button can be found. This is similar to HTML, where the BODY element will have a BgColor attribute that defines the background color of the HTML page. Detailed information about all skin elements and their attributes is included in the Skin Programming Reference section.

XML has a few simple rules that you need to know to create skins. Unlike HTML, XML requires you to follow the rules exactly.

Enclose Elements with Angle Brackets

All elements are enclosed by angle brackets; for example, the Button element is typed like this:

```xml
<BUTTON>
```

You do not need to type the word "Button" in all uppercase letters, but the convention of typing element names in all uppercase is used in the example code of this SDK.

Put Attributes Before the Closing Bracket

All attributes for a particular element must be included before the closing angle bracket. An attribute consists of the attribute name followed by an equal sign (=) and the value of the attribute in quotes.

```xml
<BUTTON image="mysong.wma">
```

You do not need to type the word "image" in lowercase, but the convention of typing attribute names in lowercase is used in the example code of this SDK. Also note that the value of the attribute is enclosed in quotation marks.

Opening and Closing Elements
Some elements are grouped together inside another element. For example, the **BUTTONGROUP** element does not make a lot of sense unless you use one or more **BUTTONELEMENT** elements with it. To make the grouping clear, you need to have an opening and closing tag for each element. The opening tag is just the element name and any related attributes, surrounded by angle brackets. The closing tag is the element name, preceded by a forward slash, (/) and then enclosed by angle brackets. For example, the **BUTTONGROUP** element opening tag looks like this:

```html
<BUTTONGROUP>
```

The closing **BUTTONGROUP** tag looks like this:

```html
</BUTTONGROUP>
```

You would put the **BUTTONELEMENT** tags between the opening and closing **BUTTONGROUP** element tags. For example:

```html
<BUTTONGROUP>
    <BUTTONELEMENT/>
    <BUTTONELEMENT/>
    <BUTTONELEMENT/>
</BUTTONGROUP>
```

**Closing Off Elements**

If an element has no other elements inside it, you must put a forward slash at the end of the element name just before the closing angle bracket. For example, in the code above, each **BUTTONELEMENT** element has a forward slash to indicate that there are no other elements nested within it.

In other words, you must either have a closing element tag or close off your element with a forward slash.

This is correct:

```html
<BUTTONGROUP>
    <BUTTONELEMENT/>
    <BUTTONELEMENT/>
</BUTTONGROUP>
```

This is not correct:

```html
<BUTTONGROUP/>
    <BUTTONELEMENT/>
    <BUTTONELEMENT/>
</BUTTONGROUP>
```

This is also not correct:

```html
<BUTTONGROUP>
    <BUTTONELEMENT>
    <BUTTONELEMENT>
</BUTTONGROUP>
```

The following section provides more information about skin definition files:

- [Skin Definition File Structure](#)
Skin Definition File Structure

The skin definition file must follow a specific structure. You start with a Theme, create one or more Views, and then define each View with the user interface elements appropriate for the type of View you want to use.

Theme

At the top level, you must start the skin definition file with the **THEME** element and close with it.

```
<THEME>
...
</THEME>
```

The **THEME** element is the root element for your skin. There can be only one **THEME** element in a skin definition file, and it must be at the top level. **THEME** elements have specific and ambient attributes, though most of the time you will not need to use them. For more information about these attributes, see the Skin Programming Reference.

View

Each Theme must have at least one View. The View governs the particular image you see on the screen. You may want to have more than one View, so you can switch back and forth. For example, you might want to have a large view for working with playlists, a medium view for watching visualizations, and a tiny view that fits in a corner of the screen.

If you are creating multiple views, you will want to make sure that each view has a unique ID attribute value that will be used to identify the view. You must define the **backgroundImage** attribute or your view will have no starting image. If you do not want to display a rectangular image, you will probably want to use the **clippingColor** attribute to define the areas of your skin that will not display, and you will probably want to set the **titleBar** attribute of the **VIEW** element.

Each **VIEW** element can also have one or more **SUBVIEW** elements. A **SUBVIEW** element is similar to a **VIEW** and can be used for parts of your skin that you want to move around, hide, or show. For example, a **SUBVIEW** element might be used to create a sliding tray that pops out of your skin to display a graphic equalizer. **SUBVIEWs** can be aligned with the **VIEW** and have other special relationships to the **VIEW**.

Initializing the Player
You can set certain initial properties of Windows Media Player by using the **PLAYER**, **SETTINGS**, and **CONTROLS** elements. For example, you could set the volume to an initial level or give a default value for a file name.

The following code shows how to set the **URL** value in a skin:

```xml
<PLAYER
    URL = "http://proseware.com/mellow.wma">
</PLAYER>
```

Note that file **URL** is used and that the slashes are escaped by another slash. If you wanted to set the **autoStart** attribute of **SETTINGS** to False, you would use the following code:

```xml
<PLAYER>
    <SETTINGS
        autoStart = "False">
    </SETTINGS>
</PLAYER>
```

Note that the **SETTINGS** element is nested inside the **PLAYER** element.

Using these elements, the following attributes and events can be specified at design time:

**PLAYER**

- url
- Buffering
- CurrentItemchange
- Currentplaylistchange
- Error
- Markerhit
- Mediacollectionchange
- Modechange
- Mpenstatechange
- Mlaylistchange
- Mlaystatechange
- Mpositionchange
- Mscriptcommand
- Mtatuschange

**SETTINGS**

- autoStart
- balance
- baseURL
- defaultFrame
- enableErrorDialogs
- invokeURLs
- mute
- playCount
- rate
- volume
CONTROLS

- currentMarker
- currentPosition

Other UI Elements

Once you have defined your THEME and VIEW elements, you must populate your VIEW with specific user interface elements. You do not have to use all the available elements in a skin, just the ones you need.

If an element can be seen by the user, it is called a control. The following controls are available for skins:

- Buttons
- Sliders, custom sliders, and progress bars
- Text boxes
- Video windows
- Visualization windows
- Playlist windows
- SubView windows

In addition, several elements can be used to further define Windows Media Player actions, but they require visual elements such as buttons or sliders:

- Video settings
- Equalizer Settings
- Visualization settings

Buttons

Buttons are the most popular part of a skin. You can use buttons to trigger actions such as play, stop, quit, minimize, and switch to different view. Windows Media Player provides the skin creator with two types of button elements: the BUTTON element and the BUTTONGROUP element. In addition, there are several predefined types of buttons.

BUTTON element

The BUTTON element is used for stand-alone buttons. If you use the BUTTON element, you must supply an image for each button and define the exact location, in pixels, where you want the button to appear, relative to the background image. One of the advantages of the BUTTON element is that you can change the button image dynamically.

BUTTONGROUP element

The BUTTONGROUP element is used for groups of buttons. In fact, you must enclose each BUTTONGROUP element with a set of BUTTONGROUP tags. Using button groups is easier than using individual buttons because you do not have to specify the exact location for each button. Instead, you supply a separate image map that defines the actions that will take place when the mouse hovers over or clicks an area on your background. Using image maps is easy because you can take the art you created for your background and copy it to a mapping layer in your art program. Using your art program is faster and more precise than trying to define exactly where a non-group button should be placed on your background.

Predefined buttons
There are several predefined buttons. For example, you can use a PLAYELEMENT button to play media files and a STOPELEMENT to stop. See **BUTTONGROUP** Element and **BUTTON** Element in the Skin Programming Reference. The **IMAGEBUTTON** can be used to display images that can change in response to specific events.

**Sliders**

Sliders are useful for working with information that changes over time. For example, you might use a slider to indicate the amount of music that has already played for a given media. They can be horizontal or vertical, linear or circular, or any shape you can think of. Sliders come in three varieties: sliders, progress bars, and custom sliders.

Sliders

You can use the **SLIDER** element for volume controls or to allow the user to move to a different part of the media content.

Progress bars

Progress bars are similar to sliders. Progress bars are designed for displaying information that changes, but not data that the user will want to interact with. For example, you might want to use a progress bar to indicate buffering progress.

Custom sliders

A custom slider is provided so you can create controls such as knobs, or do unusual control mechanisms. For example, if you want to create a volume control that wraps around your skin, you can do it with a custom slider. To set up the custom slider, you must create an image map that contains grayscale images to define the locations of the values on the slider. This is relatively easy to do with an art program that has layers.

**Text**

You can use the **TEXT** element to display text on your skin, such as song titles.

**Video**

You can display video in your skin. The **VIDEO** element allows you to set the size and position of the video window.

You can also allow the user to change the video settings with the **VIDEOSTRINGS** element. For example, you can create controls to adjust the brightness of the video.

If you do not supply a video element and the content contains video, Windows Media Player will return to Full Mode and your skin will not be displayed.

**Equalizer Settings**

You can set the filtering for specific audio frequency bands by using the **EQUALIZERSETTINGS** element. Essentially this means you can boost the bass, tweak the treble, and set up your sounds to match your ears or your living room.

**Visualizations**
You can display visualizations in your skin. Visualizations are visual effects that change over time as audio is playing through Windows Media Player. The **EFFECTS** element determines where the visualizations will play, what size the window will be, and which visualizations will be played.

**Playlists**

You can use the **PLAYLIST** element to allow the user to select an item from a specific playlist.

**SubViews**

You can use **SubView** to display secondary sets of interface controls, such as a playlist or video controls.

**See Also**

- **Skin Files**

---

**Art Files**

You must create one or more art files for your skin. Without art, the user will have nothing to look at. You could create an invisible skin, but no one would see it! And even then, you would still have to create art files to hold your invisible images, because the skin definition file requires art files for specific elements.

There are three uses of art in skins:

**Primary Images**

You must create a primary image for your skin. This is what the users will see when they install your skin. The primary image is composed of one or more images that are created by specific skin controls. If you have more than one control, you must specify the z-order. It defines which controls are displayed "in front" of other ones. Each **View** control will have a background image that you can add other element images to, allowing you to create a primary composite image. You also may have secondary images, such as a sliding tray, that do not display when your skin first appears, but that show up when the user takes some action. These follow the same rules as primary images, in that they are created with a set of controls.

**Mapping Images**

One of the most powerful features of Windows Media Player skins is that you can use image mapping to trigger events for your skin. Image maps are files that contain special images. The images in an image map file, however, are not meant to be viewed by the user, but are used by Windows Media Player to take action when the user clicks on your skin. In essence, the user cannot see them, but the mouse can. Different controls need
different kinds of image maps. For example, if you color part of an image map a specific red value, and the user clicks on the corresponding area of your primary image, a button will fire an event. Color is used to define which events are triggered by clicks in what areas of the skin. This may sound odd, but it allows a great deal of artistic control over the actions that your skin can process.

Alternate Images

You can also set up alternate images to display when a user does something. For example, you can create an alternate image of a button that will be displayed only when the mouse hovers over the button. This is a good way to let users know what they can do, and also allows for a highly discoverable user interface. By using ToolTips and hover images carefully, you can create unusual user interfaces that still give the user feedback on what options are available.

The following sections provide more information about art files:

- Primary Images
- Mapping Images
- Alternate Images
- Art File Formats
- Simple Art Example

See Also

- Skin Files

Primary Images

You will want to create primary images for your skin. The primary image is composed of one background image defined by the VIEW element, and any other images that go with specific elements. For example, if you create a BUTTON element, you must include the art for that button, and be sure that it has a higher z-order than the background image.

Creating Art

There are numerous computer applications that will help you create artistic masterpieces, and more are appearing every day. Many people will create skins as part of a team. There are three areas of skin creation that relate to art. You may be one or more of the following creators:

Illustrator

You are an illustrator, fine artist, or painter, and you use programs focused on creating art from scratch, such as
Adobe Illustrator and Metacreations Painter. You can draw and paint original images. These images can be used to create original skin art.

Designer

You are a commercial artist, designer, or art director, and you take advantage of programs that will do a lot of the art modification for you, such as Adobe Photoshop or Jasc PaintShopPro. If you are on a team, someone else may do the original art or you may modify existing art that you obtained through a scanner, clip art collection, or digital camera.

Programmer

You do not create or modify art, but you create the code that uses the art. You may help the illustrator or designer understand the requirements for the art.

The creation process is fluid and the work will probably go back and forth between team members as the skin evolves. You might start out with an image created by an illustrator, the programmer may request image maps, and the art director may make modifications to the art based on technical suggestions from the programmer.

Similar types of teams have evolved for games and Web pages. It is unusual to have someone with all three skills, since each skill requires experience and training. All that matters is that you identify the skills needed and find the person to do the job, even if it turns out to be you for everything!

Mapping Images

Mapping images are used for specific controls to specify which regions will respond to mouse clicks and to determine which controls receive which events. The mapping image must be the same height and width as the primary image and should be registered exactly so that the images in both files are lined up exactly. For this reason, an art program such as Adobe Photoshop is recommended because you can work in layers to be certain your images are perfectly aligned.

Different controls require different types of mapping. A ButtonElement control needs a map that has a different color for each button. The colored areas in the mapping file must correspond to the areas of the buttons you want to map.
Alternate Images

Alternate images are images that are not seen when the skin is initially displayed, but become visible in response to mouse events. For example, you can set up the skin so that when a mouse hovers over a specific area, the art in that area will change. This can inform users visually that they can click on an area to do something. You might also use an alternate image when the user clicks on an area to give them visual feedback that the click was received by Windows Media Player.

Art File Formats

The following art file formats are recognized by Windows Media Player for skins:

**BMP**

Microsoft Windows Bitmap images are recommended because they offer the most control over the exact image and colors.

**JPEG**

Compressed image format used for Web pages. JPEG format files usually have .jpg file name extensions.

**GIF**

Compressed image format used for Web pages. Animated GIFs are supported.

**PNG**

Compressed image format used for Web pages.

**Note** If you use one of the compressed file formats that defines a color as transparent to a Web browser, do not define a color as transparent in the image file. Use a visible color to represent transparent areas in your image, and then define that color as transparent in the skin definition file. For example, if you create a gif file with some areas transparent, they will not be transparent in your final image and you will not be able to use the color you set as transparent in your gif file for the transparency color in your skin.
Simple Art Example

Three art files are needed to create a simple skin with two buttons. A primary image and a mapping image are required, and an alternate image provides a visual cue to the user that a button is clickable.

These art files were created in Adobe Photoshop but any art program could be used. An art program that uses layers is easier to work with because you will want to make sure that your primary, mapping, and alternate images all are the same size and line up with each other.

The detailed instructions on creating the art are in the Skin Creation Guide.

Primary Image

The primary image is a simple yellow oval with two buttons, a pink one to start Windows Media Player and purple button to stop it. The background is a slightly darker yellow than the oval.

![Primary Image](image)

The primary image was created in Adobe Photoshop from the following images, each in a separate layer. First an oval was created with a layer bevel and emboss effect.

![Oval Image](image)

Then the two buttons were created, also with layer bevel and emboss effects.

![Button Image](image)
Next the image background was created. A slightly darker yellow was chosen so that any anti-aliasing between the oval and the background will not be noticed. The color value is #CCCC00. This number is shown in the Photoshop Color Picker dialog box.

The layers that contained these images were made visible and saved as a copy in the BMP format, creating the primary image. The primary composite image will be used by the `backgroundImage` attribute of the `VIEW` element.

**Mapping Image**

A mapping image is needed to specify when and where a skin is clicked. A mapping image was created with a red area and a green area.

The green area will be used to identify the area on the skin that will start Windows Media Player and the red area will be used to stop it. The mapping image is the same size as the primary image.

The mapping image was created by copying the button layer to a new layer and turning off the bevel and emboss effect. Flat images are needed for mapping because Windows Media Player will be looking for single color values in each area. It can only search for a color you define, for instance red (#FF0000), and if your image has a bevel or other effect, not all of it will be the exact red you need. To make the mapping buttons an easy color to remember, the images were filled with pure red and pure green, but any color can be used. You will need to remember the color numbers in your map so that they can be entered in the XML skin definition file. In this case, red is #FF0000 and green is #00FF00.

Then, with only the new layer visible, the image was saved as a copy to a BMP file. It will be called by the `mappingImage` attribute of the `BUTTONGROUP` element.

**Alternate Image**

Alternate images are not required but are very useful to give visual cues to the user. In this case, a hover image is recommended so that the user knows what areas can be clicked on.

An alternate image was created with two yellow buttons.
The alternate image was created by copying the original button layer to a new layer and then changing the fill color to yellow. The bevel and emboss effect was kept. Then a new layer was created and images were added: the arrow indicates "play" and the square indicates "stop". Then, with only the new yellow button and type layers visible, the image was saved as a copy to a bitmap file.

The result is that when the mouse hovers over an area defined by the mapping image, the hover image will be displayed, alerting the reader that if they click on that spot, they can play or stop Windows Media Player.

**Final Image**

Here is the final image of the skin:

![Final Image](image)

And this is the image you will see if you hover over the pink button on the right:

![Hover Image](image)

**XML Code for the Art Example**

The details of how to write XML code are given in the [Skin Creation Guide](#), but to show how little code is needed to create a working skin, here is the code for the artwork in this example. Predefined buttons are used for the play and stop functions. You must load a file or playlist from the Windows Media anchor. When Windows Media Player view shifts to skin mode, a small box appears in the lower right hand corner of the screen. This box is called the *anchor* and clicking on it gives you the minimum functionality needed, in case a skin does not provide a way to return to full mode. The user can switch between modes using the *View* menu if in full mode or the anchor if in skin mode.

```xml
<THEME>
  <VIEW
    clippingColor = "#CCCC00"
    backgroundImage = "background.bmp"
    titleBar = "false">
    <BUTTONGROUP
      mappingImage = "map.bmp"
      hoverImage = "hover.bmp">
      <PLAYELEMENT
        mappingColor = "#00FF00"/>
      <STOPELEMENT
        mappingColor = "#FF0000"/>
    </BUTTONGROUP>
  </VIEW>
</THEME>
```
JScript Files

JScript files are loaded with the `scriptFile` attribute of the `VIEW` element. They must be text files and should use the file extension .js. If you have a JScript file that has the same name as the skin definition file, the JScript file will be loaded at the same time as the skin definition file. For example, if you have a skin definition file named laure.wms, and you have a JScript file called laure.js, the laure.js file will be automatically loaded.

You can use Microsoft JScript to create elaborate functionality behind your skin. By creating functions in JScript, you can do almost anything imaginable with skins. For example, you could use a different playlist for every day of the week, but always have the same one on Friday. See Using JScript for more information about using JScript with skins. Note that VBScript, which can be used when embedding the Windows Media Player control in a Web page, is not supported for use with skins.

See Also

- Skin Files

Writing Code

If you create a skin using only elements and attributes, all you will have is a pretty picture. The real power of skins comes from being able to respond to events. You want your skin to be able to do something when the user clicks on a button, and you will also want to respond to changes that happen to Windows Media Player, such as the progress of the media file that is playing.

The following sections discuss writing code for skins:
Handling Events

Aside from the XML code that you write to initialize attributes for your skin, the primary code you write is JScript code to handle events. Events are either external or internal; that is, they come from the user or from Windows Media Player.

The following sections provide details about handling events:

- External Events
- Internal Events
- Writing Event Code
- Secondary Events

See Also

- Writing Code
External Events

When users click a button or press a key, you can respond to their input with event handlers. An event handler is a section of code that runs whenever the event is triggered.

The following events are supported by skin elements:

- load
- close
- resize
- timer
- click
- dblclick
- error
- mousedown
- mouseup
- mousemove
- mouseover
- mouseout
- keypress
- keydown
- keyup

See the Skin Programming Reference for more details about specific events.

A typical external event handler would name the event and define the code that will run. For example, if you want to create code to start Windows Media Player when the user clicks on a button, you would put the following line in your button code.

```
onclick = "JScript: player.URL = 'http://proseware.com/laure.wma' ; "
```

This will play the file named laure.wma. Note that you add the word "on" to specific events.

See Also

- Handling Events

Internal Events

You can detect changes that occur in Windows Media Player or changes in your own skin. These can be changes in Windows Media Player object properties or methods, changes in skin attributes, and so on.
Player Property Changes

You can process changes in Windows Media Player by using the wmpprop listener. You must set up the listener as a value of an attribute. Put the value in double quotes, and start with the word "wmpprop" followed by a colon. Then you include the property you want to listen to. When the property changes, the value of the attribute will change also. For example, to have a slider element value change whenever the value of the `currentPosition` attribute changes, type the following:

```
<SLIDER id="mySlider" value="wmpprop:player.Controls.currentPosition" />
```

**Important** Do not use wmpprop on Windows Media Player methods. Unexpected results may occur.

Windows Media Player Method Changes

You can make your skin respond to the availability of methods on Windows Media Player using wmpenabled and wmpdisabled. These are used similarly to the wmpprop listener except that you can only use these on methods of the Control object that are supported by the `isAvailable` method.

For example, you could enable a button only when the Play method is enabled, using code like this:

```
<BUTTON ... enabled="wmpenabled:player.Controls.Play();" />
```

**Important** Do not use wmpenabled or wmpdisabled on Windows Media Player properties. Unexpected results may occur.

Skin Attribute Changes

You can respond to changes in your skin attributes in one of two ways, by using wmpprop or the `_onchange` event.

You can use wmpprop to listen for changes in your own skin. For example to show the Slider value in a text box, you could type the following:

```
<TEXT ... value="wmpprop:mySlider.value">
```

You can use the `_onchange` event to process events inside an element. You must attach the name of the attribute you want to track to `_onchange`. For example if you want to track the value of a text box, you would type:

```
value_onchange
```

and then assign a JScript string that you want to run when the value changes. For example, to respond to a change in the value of a text box that can be used to adjust the volume of Windows Media Player, type the following inside your Text control as an attribute:

```
value_onchange = "JScript: player.Settings.Volume = myText.value"
```

See Also

- Handling Events
Writing Event Code

Events are treated similarly to attributes. You must give the event a value, and the value is the code you want to run when the event happens. The word "on" is added to the front of the event name; for example, the click event will become `onclick`.

The event value is in double quotes and starts with the word JScript followed by a colon. The code you want to run comes next, followed by a semicolon and the closing double quotes. For example, to stop playing when the user clicks on a button, type the following as an attribute in your `BUTTON` element code:

```javascript
onclick = "JScript: player.Controls.Stop() ; "
```

If you have a code that requires quotes, use single quotes. Care must be taken when using quotation marks so that they are balanced properly. Here is an example of using both types:

```javascript
onclick = "JScript: player.URL = 'http://proseware.com/laure.wma' ; "
```

You can also change attributes of your skin when handling an external event. For example, to close a view named `myView`, you could type:

```javascript
onclick = "JScript: myView.close() ; "
```

See Also

- [Handling Events](#)

Secondary Events

You can determine what other events are taking place when a specific event is triggered. For example, when a mouse button is clicked, you may want to know whether the ALT key was down at the same time.
Event Attributes

The following event attributes are supported for skins:

- altKey
- button
- clientX
- clientY
- ctrlKey
- fromElement
- offsetX
- offsetY
- screenX
- screenY
- shiftKey
- srcElement
- toElement
- x
- y
- keyCode

For more information about these attributes, see the Skin Programming Reference.

Using Secondary Events

You can only process event attributes in JScript code. You must use the following syntax:

```javascript
event.eventattributename
```

where eventattributename is the name of the event attribute. For example, to determine whether the ALT key was pressed during a click event, you could use the following lines in your JScript code:

```javascript
wasAlt = event.altKey;
if (wasAlt = true)
{
    // Do something here.
}
```

See Also

- [Handling Events](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Using JScript

Windows Media Player requires Microsoft Internet Explorer version 4 or later. When writing code, do not assume that the user's machine will contain a version of Microsoft JScript later than 4.0.

The following sections provide details about using JScript in skins:

- Working with the Player
- Event Handlers
- Calling Functions

See Also

- Writing Code

© 2000-2003 Microsoft Corporation. All rights reserved.

Working with the Player

When using Microsoft JScript to access methods and properties of Windows Media Player, you must use the name "player" for the name of the control. For example, to reference the Stop method, you must type:

    player.Controls.Stop()

The player global attribute is the key to accessing the Windows Media Player control through skin scripting. Through this attribute, all the objects of the Windows Media Player control become accessible for run time modification through their properties and methods. Additionally, the PLAYER element is available in order to specify event handlers and the url attribute at design time.

See Also

- Using JScript
Event Handlers

Microsoft JScript is used to process events in the skin definition file. See Handling Events for more information about event handlers.

You can have more than one line of code in an event handler, but care must be taken not to exceed the line length that JScript permits. Separate the lines by semicolons.

```javascript
onclick = "JScript: player.URL = 'http://proseware.com/cool.wma' ; myText.value = 'Playing
```

See Also

- Using JScript

Calling Functions

If you need to call more than a few lines of code, you can load a script file using the `scriptFile` attribute of the `VIEW` element, and call the functions in the script. You can load more than one file per view; for example:

```javascript
scriptFile = "myfile1.js;myfile2.js;myfile3.js"
```

After the script files are loaded, you can call functions in them from inside your View section. For example, to call a function called `myfunction` when something is clicked, type:

```javascript
onclick = "JScript: myfunction()"
```

**Note** If you have more than one view, only the functions in files loaded into the current view are available to the XML and JScript code running with the current view. Files loaded in other views are not in scope with the current view.

See Also

- Using JScript
Debugging Code

You often will want to see what is happening inside your skin. You can do this through a text control or through a log file.

You can create a TEXT element and place it on a part of your skin temporarily. For example, you could use the following code to create your TEXT element:

```xml
<!-- debugging control -- remove later -->
<TEXT
    id = "debug"
    foregroundColor = "white"
    backgroundColor = "black"
    value = "debug"
    top = "100"
    left = "50"
    height = "15"
    width = "100"
    z-order = "5" />
<!-- end debugging control -->
```

Then, for example, if you want to see the current position of the media content in Windows Media Player, you could use code similar to the following to display the current position in the text box.

```xml
<Player
    id = "myplayer">
    <Controls
        id = "mycontrols"
        currentPosition_onchange="value=player.controls.currentPosition"/>
</Player>
```

See Also

- About Skins
Submitting Your Skin

Once you've created and tested your skin, you may want to distribute it to the world. Make sure that you are not violating any copyrights and use only original art or art that does not have a copyright. Before you distribute your skin, ask everyone you know to try out your skin first. Think of all the different ways people can interact with your skin. You want to find and fix any problems before you send it out to the world.

Consult the Windows Media Web Gallery for more information on how to distribute your skin.

See Also

- About Skins

Referencing Skins in URLs

If you use a URL to load a media file that will be played by Windows Media Player, you can request that a particular skin be used with that file. If the skin is already installed on the user's machine, it will be used; otherwise the previous skin will be used.

To request a skin, add the following to the end of the URL:

```
?WMPSkin=skinname
```

where *skinname* is the name of the skin you want to request. Do not use quotes around the name of the skin.

For example, to request the headspace skin be used, type the following http URL:

```
http://www.proseware.com/mymedia.wma?WMPSkin=headspace
```

See Also

- About Skins
New for Windows Media Player 9 Series

Windows Media Player 7.0 introduced skin technology. The available elements and attributes remained unchanged for Windows Media Player 7.1 and Windows Media Player for Windows XP. Windows Media Player 9 Series introduces new elements and attributes you can use. Each topic in the Skin Programming Reference section includes a Requirements section that lists the minimum requirement for the individual element, method, event handler, or attribute. The following list also details the new items for particular versions of Windows Media Player.

Added for Windows Media Player for Windows XP

- EFFECTS.effectCanGoFullScreen
- EFFECTS.effectCount

Added for Windows Media Player 9 Series

- AmbientAttributes.accDescription Attribute
- AmbientAttributes.accKeyboardShortcut Attribute
- AmbientAttributes.accName Attribute
- AmbientAttributes.alphaBlend Attribute
- AmbientAttributes.alphaBlendTo Method
- AUTOMENU Element
- BALANCESLIDER Predefined Element
- BUTTONELEMENT.click Method
- BUTTONELEMENT.index Attribute
- BUTTONGROUP.buttonCount Attribute
- BUTTONGROUP.click Method
- BUTTONGROUP.getButton Method
- BUTTONGROUP.hueShift Attribute
- BUTTONGROUP.saturation Attribute
- CLOSEBUTTON Predefined Element
- COLUMN Element
- CURRENTPOSITIONTEXT Predefined Element
- DROPDOWNPLAYLIST Predefined Element
- DURATIONTEXT Predefined Element
- EDITBOX Element
- EFFECTS.effectTitle Method
- EFFECTS.effectType Method
- EQUALIZERSETTINGS.crossFade Attribute
- EQUALIZERSETTINGS.crossFadeWindow Attribute
● **EQUALIZERSETTINGS.currentSpeakerName** Attribute
● **EQUALIZERSETTINGS.enableSplineTension** Attribute
● **EQUALIZERSETTINGS.enhancedAudio** Attribute
● **EQUALIZERSETTINGS.normalization** Attribute
● **EQUALIZERSETTINGS.normalizationAverage** Attribute
● **EQUALIZERSETTINGS.normalizationPeak** Attribute
● **EQUALIZERSETTINGS.presetTitle** Method
● **EQUALIZERSETTINGS.speakerSize** Attribute
● **EQUALIZERSETTINGS.splineTension** Attribute
● **EQUALIZERSETTINGS.truBassLevel** Attribute
● **EQUALIZERSETTINGS.wowLevel** Attribute

● **ITEM** Element
  ● **ITEMSPLAYLIST** Predefined Element
  ● **LISTBOX** Element
  ● **MINIMIZEBUTTON** Predefined Element
  ● **MUTEBUTTON** Predefined Element
  ● onendalphablend Event Handler
  ● **PLAYLIST.dropDownBackgroundImage** Attribute
  ● **PLAYLIST.dropDownImage** Attribute
  ● **PLAYLIST.editButtonVisible** Attribute
  ● **PLAYLIST.getNextCheckedItem2** Method
  ● **PLAYLIST.getNextSelectedItem2** Method
  ● **PLAYLIST.hueShift** Attribute
  ● **PLAYLIST.itemCount** Attribute
  ● **PLAYLIST.itemErrorColor** Attribute
  ● **PLAYLIST.itemMedia** Attribute
  ● **PLAYLIST.itemPlaylist** Attribute
  ● **PLAYLIST.leftStatus** Attribute
  ● **PLAYLIST.rightStatus** Attribute
  ● **PLAYLIST.saturation** Attribute
  ● **PLAYLIST.setCheckedState2** Method
  ● **PLAYLIST.setSelectedState2** Method
  ● **PLAYLIST.statusColor** Attribute
  ● **REPEATBUTTON** Predefined Element
  ● **RETURNBUTTON** Predefined Element
  ● **SEEKSLIDER** Predefined Element
  ● **SHUFFLEBUTTON** Predefined Element
  ● **STATUSTEXT** Predefined Element
  ● **THEME.playSound** Method
  ● **TRACKNAMETEXT** Predefined Element
  ● **VIEW.backgroundImageHueShift** Attribute
  ● **VIEW.backgroundImageSaturation** Attribute
  ● **VIEW.resizeBackgroundImage** Attribute
  ● **VOLUMESLIDER** Predefined Element
  ● **WMPEFFECTS** Predefined Element
  ● **WMPVIDEO** Predefined Element

See Also

● About Skins
● Skin Programming Reference
● What's New for Windows Media Player Skins
Borders for Windows Media Player

A border is similar to a skin, but instead of replacing the user interface for the compact mode of Windows Media Player, a border is embedded in the Now Playing pane of the full mode Windows Media Player.

By using the Windows Media Download file format (WMD), you can include content with a border, paving the way to multimedia applications.

A sample WMD file that includes a border and embedded content is included in the samples directory of this SDK.

Creating a Border

A border is created the same way as a skin. The only difference is that the skin is embedded inside the Now Playing pane. This means that the skin size cannot be calculated and that all skin elements must be relative. If the end user resizes Windows Media Player, portions of the border may be clipped off and not seen.

Loading a Border

A border is loaded when a Windows Media metafile that uses the SKIN element is loaded. The HREF attribute of the SKIN element must reference a skin. A typical SKIN element would look like this:

<SKIN HREF="myborder.wmz">

For more information, see SKIN Element.

Including Content with a Border

You can include content with a border by using a Windows Media Download file (with a .wmd file name extension). Follow these steps:

1. Create the border. Take care to create it in such a way that resizing will not ruin the composition of the border. For example, do not include a background file; make the background transparent so that a visualization could run behind it.
2. Compress the skin contents (art, JScript files, and skin definition file) into a file with a .wmz file name extension.
3. Create a Windows Media metafile (with a .asx file name extension) that references the compressed border file (with a .wmz file name extension). The metafile can include playlist information with metadata for art and URLs for specific content.
4. Assemble the digital media content.
5. Compress the border, metafile, and content into one file and give it a .wmd file name extension.

Using a Border

After you have created the Windows Media Download file, all that the end user has to do is double-click on it. The file will be downloaded to the end user's computer. The files inside the package will be unpacked, the playlist will be added to the playlists, the border will appear in the Now Playing pane of the full mode Windows Media Player, and the first item on the new playlist will begin playing.

See Also

- About Skins
- Samples
- Windows Media Download Packages

Skin Creation Guide

This guide is a series of detailed explanations of how to create different kinds of skins. For more general information on skins, see About Skins. For specific details about every attribute, method, and event used in skins, see the Skin Programming Reference. As you get more involved in the programming of your skin, you may want to read the part of this SDK covering the Windows Media Player Object Model.

In this guide, instructions for creating the art will be given for Adobe Photoshop 5.5, a popular art manipulation program. The specific instructions may be different if you have a similar art program such as Jasc Paint Shop Pro or Sonic Foundry Viscosity, but the concepts will be the same. Photoshop was chosen for two reasons: it is a popular art program for commercial artists, and it works with layers. As you will see in the tutorials, layers are very useful for skin creation.

This guide will cover the following tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Your First Skin</td>
<td>A walk-through of a simple skin that starts and stops Windows Media Player.</td>
</tr>
<tr>
<td>Adding a Playlist</td>
<td>How to use a simple playlist.</td>
</tr>
<tr>
<td>Choosing Files</td>
<td>How to pick files with an open file dialog box.</td>
</tr>
<tr>
<td>Adding Video</td>
<td>How to put a video window into your skin.</td>
</tr>
</tbody>
</table>
Building Your First Skin

As explained in greater detail in About Skins, a skin is defined in a skin definition file, and requires one or more supporting art files, Microsoft JScript files, and digital media files. This tutorial for a very simple skin requires only the skin definition file and three art files. You will go through four steps described in the following topics:

- Creating the Primary Art File
- Creating the Mapping File
- Creating the Hover Image
- Creating the Skin Definition File

See Also

- Skin Creation Guide
Creating the Primary Art File

The primary art file will contain the art that the user of your skin first sees. In this case, you will be creating images in different layers of your art program. The reason for using layers is that you will copy specific layers later to create map files and alternate art files.

To create the primary art file, you will create the following layers in the following order:

**Skin background layer**

This is the color that will be transparent when the skin is displayed. Create a layer for this first, but choose the final color of this layer after you chose a color for the skin container layer. This color should be similar to, but not the same as, the skin container layer, to hide any anti-aliasing effects.

**Skin container layer**

This is the image that will form the outline of your skin and will be what the user sees. It also will be the container for the two buttons in this example. Think of your skin as a container for user interface controls such as buttons, sliders, and so on. In this example, the container is a yellow oval.

**Play and Close button layers**

These are the two user interface controls that this example uses. You will put them in separate layers so that you can easily adjust them or copy them later.

Before you create your layers, you must create the file that will hold your layers. Start up Photoshop and create a new file that is 100 pixels high and 200 pixels wide. The file used to create the art for this sample is called tiny.psd and is included with the SDK.

All instructions are given in terms of Photoshop, but any other art program can be used to create skins as long as you can save to one of the file formats supported by the Windows Media Player (BMP, GIF, JPG, and PNG). You will find skin creation easier if you use an art program that has layers, such as Adobe Photoshop, Jasc Paint Shop Pro, or Jedor Viscosity. Layers are extremely useful because images must be properly aligned for image mapping and display of alternative images.

**Skin Background Layer**

Create a new layer and name it *Skin background*. This will become the transparency color you will define in the skin definition file. Wait until the color for the skin container is chosen before filling the skin background layer with a specific color.

**Skin Container Layer**

Next create a new layer and call it *Skin container*. This will define the edges of your skin and will be the container for the buttons.

Choose a foreground color for the shape, using the Web color sliders. In this example, the color 
"#DBDD11" was chosen.

Next create an oval shape. The easiest way is to use the Elliptical Marquee tool and create an oval selection. When you have created an oval selection that is the size and shape you want, fill the selection with the foreground color and cancel the selection.
Finally, to make this look a bit more interesting, apply the layer effect of Bevel and Emboss with the default values.

Your skin container layer should look like this:

![Skin Container](image)

**Background Skin Color**

Now that you have chosen a foreground color for your skin container shape, you can choose a similar color for your skin background layer. You do not want the exact same color, or your skin container will be transparent also. In fact, be sure you do not use this exact color anywhere else in your skin, even in photographs, because wherever this color appears, the desktop image will appear instead.

You want a color close to the skin container color to avoid anti-aliasing effects. For example, if you have a black background, some bits of black may show up around the edge of your skin. By choosing a color close to the color of the skin container, any stray pixels that show up in the anti-aliasing process will be unnoticed.

Anti-aliasing is the process of smoothing the edges of slanted or curved shapes. Anti-aliasing creates new colors, for pixels along the edges of a shape, that are a blend of the foreground color and the background color. Some of these in-between colors can cause pixels to be missed when the background color is made transparent.

Your skin background layer should look like this:

![Skin Background](image)

**Play and Close Button Layers**

Create a new layer and name it *Close button*. Using the Eliptical Marquee selection tool again, create a circle and position it on the left side of the overall image. Turn on the visibility of the skin container file to help place the selection.

When you are satisfied with the placement, fill the selection with any color (except the color of the skin container or the skin background). In this example, a purple color was chosen. You do not need to remember the number of the color. Then cancel the selection and apply another default Bevel and Emboss layer effect. If you want to apply non-layer effects to your button, make a copy of the original for later use in mapping.

Your Close button should look like this:
Create a new layer and name it *Play button*. Use the same techniques you did for the Close button, but give it a different color. In this case, a pink button color was chosen, but any color can be used as long as it is not the same color as the skin container (because it would blend into the container) or the skin background color (because it would become transparent).

Your Play button should look like this:

![Play button image]

**Combine Layers and Save**

You are now ready to create the primary art file. Hide all layers and then show only the following layers, in this order (top to bottom):

- Play button
- Close button
- Skin container
- Skin background

Save to a new file using the Save a Copy command from the File menu. Select the BMP option in the Save As portion of the Save a Copy dialog box and type a file name that you will refer to later in your skin definition file. Ideally you should save this in the same directory as your skin definition file. For example, you could call this background.bmp. Choose the default settings and save the file.

Your primary art file should look like this:

![Primary art file image]

You will use this file name as the value for the `backgroundImage` attribute of the `VIEW` element in your skin definition file.

**See Also**

- [Building Your First Skin](#)
Creating the Mapping File

Once you have created the pieces of your primary art file, it is relatively easy to create a mapping file. You will create the new mapping file by combining the art from the two button layers you already created.

1. You will need to take the two buttons you created for the primary art file and copy them to a new layer. Use the following steps: Copy the Close button layer, remove any Layer effects, and rename it Close map. The art should look flat, with no bevels.
2. Use the Color Picker to create a foreground color of pure red. Be sure the color number value is "#FF0000". Then use the Paint Bucket tool to fill the inside of the circle of the Close map layer.
3. Copy the Play button layer, remove any Layer effects, and rename it Play map. Again, the art should look flat. You do not want any effects in the mapping layer because you are just defining regions of the bitmap that Windows Media Player will use to determine where the mouse performs an action and what you want to do with it.
4. Use the Color Picker to create a foreground color of pure green. Be sure the color number value is "#00FF00". Then use the Paint Bucket tool to fill the inside of the circle of the Play map layer.

You are now ready to create the mapping art file. Hide all layers, and then show only the following layers, in this order (top to bottom):

Play map

Close map

Save to a new file using the Save a Copy command from the File menu. Select the BMP option in the Save As portion of the Save a Copy dialog box and type a file name that you will refer to later in your skin definition file. Ideally it should be in the same directory as your skin definition file. For example, you could call this file map.bmp. Choose the default settings and save the file.

Your mapping file should look like this:

As you might guess, the green area will be used to determine when to make Windows Media Player start and the red area is for telling it to stop. Any two colors can be used, as long as you use the color numbers when you
set up the skin definition file. Be sure the colors in the map are pure colors for the region you want to use and have distinct edges. A pure color would be one where every single pixel in the area has the same color value. Using an effect may blur or distort the edge, thereby slightly modifying the colors of some of the pixels. The mapping file is only seen by the mouse, not a human, so do not bother decorating it, and remove any layer effects you may have carried over from other layers.

When you save your file, the file name you choose will later be used as the value for the `mappingImage` attribute of the `BUTTONGROUP` element in your skin definition file.

See Also

- Building Your First Skin

Creating the Hover Image

The primary image of this skin is two buttons sitting on an oval. To give the user a clue on what to do, you can add hover images. These are alternate images that are displayed when the user hovers a mouse over a button. The hover buttons will also contain the play and stop VCR control symbols so that users will know exactly what they can do. Using hover images allows you to create complex, self-documenting, artistic skins.

To create the hover image, you will need to take the two buttons you created for the primary art file, copy them to new layers, and add further layers for the text. Use the following steps:

1. Copy the Close button layer and rename it `Close hover`.
2. Use the Color Picker to create a foreground color of a light yellow (#CCFF33). This was chosen to contrast with the button colors. Then use the Paint Bucket tool to fill the inside of the circle in the Close hover layer.
3. Copy the Play button layer and rename it `Play hover`.
4. Use the Paint Bucket tool to fill the inside of the circle in the Play hover layer with the same color as the Close hover circle.
5. Create a new layer and name it `Close square`. Use the Color Picker to create a foreground color of dark blue. Use the pen tool to draw a square, turn it into a selection, fill it, and delete the path. Using the Move tool, move the square and center it over the Close hover button.
6. Create a new layer and name it `Play triangle`. Use the pen tool to create the triangle for "Play" using the same techniques you did to create the Close square layer. Center it over the Play hover button.

You are now ready to create the hover art file. Hide all layers, and then show only the following layers, in this order (top to bottom):

Play triangle
Save to a new file using the Save a Copy command from the File menu. Select the BMP option in the Save As portion of the Save a Copy dialog box, and type a file name that you will refer to later in your skin definition file. Ideally you should save this in the same directory as your skin definition file. For example, you could call this hover.bmp. Choose the default settings and save the file.

Your hover art file should look like this:

![Hover Images]

The yellow hover button will show in place of the normal button. If you hover over the right button in your skin, the yellow button labeled "Play" will appear, and if you hover over the left button, the user will see "Close". The two hover images will never both be displayed at the same time, because the mouse cannot hover over both buttons at the same time. You must turn hovering on and have a hover art file that matches the areas of the mapping files to areas of the hover file.

When you save your file, the file name you choose will later be used as the value for the `hoverImage` attribute of the `BUTTONGROUP` element in your skin definition file.

See Also

- Building Your First Skin
remember to save as text every time you save.

The following sections will guide you in creating a skin definition file:

- **Skin Definition File XML Structure**
- **Start with THEME and VIEW**
- **Adding BUTTONGROUP**
- **Adding the Play BUTTONELEMENT**
- **Adding the Close BUTTONELEMENT**
- **Complete Code for Simple Skin**

**See Also**

- **Building Your First Skin**

---

**Skin Definition File XML Structure**

The skin definition file is written in XML. One of the important features of XML is that it is completely structured, and is similar to an outline. The XML code is simply a series of elements such as **VIEW** and **BUTTONGROUP**. You will start with the elements and then define them with attributes. The rest of this tutorial will give you details on the attributes, but here is the outline of the elements that will be used:

```xml
<THEME>
  <VIEW>
    <BUTTONGROUP>
      <BUTTONELEMENT/>
      <BUTTONELEMENT/>
    </BUTTONGROUP>
  </VIEW>
</THEME>
```

By keeping in mind the simple structure of the elements, you can make sense of the attributes that make each element unique. Details of each element will be covered in the remaining topics of this section. For more information about elements and attributes, see the **Skin Programming Reference**.

**See Also**

- **Creating the Skin Definition File**
Start with THEME and VIEW

Every skin must have exactly one THEME element and at least one VIEW element.

Using your text editor, create the following text:

```
<THEME>
  <VIEW
    clippingColor = "#CCCC00"
    backgroundImage = "background.bmp"
    titleBar = "False">

  </VIEW>
</THEME>
```

Leave some blank lines before the closing VIEW tag because you'll be adding more code here later.

Save your file with any file name you wish, but be sure that the extension is .wms. For example, a typical file name might be skinone.wms.

Every skin must start with <THEME> and end with </THEME>. You can only have one THEME element in your skin, but you must have one.

You must also have at least one VIEW element. You can have more than one VIEW, but this example only has one. You must have an opening <VIEW> and a closing <VIEW>. Notice that the opening </VIEW> tag does not close the tag right away, but includes several attributes before the closing angle bracket (>). The following attributes are used in the THEME element in this example:

- clippingColor

You will not always need the clippingColor attribute if the edges of your skin are rectangular. The skin in this example is oval-shaped, so you need a clipping color for the parts of the skin that you want to see the desktop through; essentially all parts outside the oval. In this example skin, we will use a dark yellow, specified as "#CCCC00" in Web format. The reasons for this choice are given in Creating the Primary Art File. Essentially, this value will always be a number that you get from your art program.

- backgroundImage

This is the name of the primary art file. It should be the exact file name and path of your primary art file. Only BMP, JPG, GIF, and PNG files are supported, and BMP is recommended.

- titleBar

This skin does not have a titleBar, so the value will be "false". You will only want a title bar if you want your
skin to have a background color and be rectangular.

Be sure that you put the closing angle bracket (>) after the titleBar value to indicate that you are finished defining the VIEW. Leave a few blank lines before the closing VIEW and THEME tags. You will need the lines for code that you will add later.

See Also

- Creating the Skin Definition File

Adding BUTTONGROUP

This example uses the BUTTONGROUP element for the coding in the skin definition file. BUTTONGROUP creates an easy way to process mouse events without having to calculate exact locations on the screen and uses color instead of x and y-coordinates.

First you must add the BUTTONGROUP tags to the skin definition file you created. Put them after the THEME tag attributes.

```xml
<BUTTONGROUP
    mappingImage = "map.bmp"
    hoverImage = "hover.bmp">
</BUTTONGROUP>
```

Leave a few blank lines above the closing BUTTONGROUP tag for the buttons you will add next.

The following attributes are used to define BUTTONGROUP:

**mappingImage**

This is the file name of the mapping art file you created before, the one with the red and green circles. This attribute is required for any BUTTONGROUP.

**hoverImage**

This is the file name of the hover art file you created before, the one with the two yellow buttons that read "Play" and "Close". This is not required, but a hover image helps to provide feedback to the user.

See Also
Adding the Play BUTTONELEMENT

Finally, you can add the BUTTONELEMENT elements that connect the visual buttons on the screen to Windows Media Player actions. This is the core of your skin and you can think of it as wiring the surface of the skin to the inner machinery of Windows Media Player.

BUTTONELEMENTs are contained within a BUTTONGROUP. You must always have at least one BUTTONELEMENT inside each BUTTONGROUP.

Put the Play BUTTONELEMENT code after the closing angle bracket of BUTTONGROUP.

```xml
<BUTTONELEMENT
    mappingColor = "#00FF00"
    upToolTip = "Play"
    onClick = "JScript: player.URL='http://proseware.com/laure.wma';" />
```

The following attributes are used to define the BUTTONELEMENT for the Play button:

**mappingColor**

This is the color value of a region in the mapping art file you created before. In this case it is the solid green color. This attribute is required for any BUTTONELEMENT. By defining this color, you are telling Windows Media Player to associate this color area with the XML code of this button.

**upToolTip**

This defines the text that will be displayed if the user hovers the mouse over the button. Do not confuse this with the hover art that will be displayed. A ToolTip is a small balloon caption that takes a moment to appear. The hover art image, however, will appear instantly in whatever color and shape you choose.

**onClick**

This defines the event that occurs when the mouse clicks on the button. The value of this event attribute is called an event handler and will be either a line of Microsoft JScript code, or a JScript function in an external text file that is loaded by the loadScript attribute of a VIEW. In this case, the JScript code calls the URL method of Windows Media Player, which loads and starts playing a file named "laure.wma". Note that the line ends with a semicolon inside the quotes, which is good JScript coding practice. Note also the use of single quotes inside the double quotes to set off the file name. For more information about JScript, see Using JScript in the About Skins section of this SDK.
Notice that there is no ending **BUTTONELEMENT** tag. If an element does not enclose another element, you can close it off with the forward slash just before the closing angle bracket. This tells XML that you are finished with that element. For example,

```xml
<BUTTONELEMENT></BUTTONELEMENT>
```

and

```xml
<BUTTONELEMENT/>
```

convey the same information in XML.

The power of skins comes from using event handlers. If the user does something with a mouse, you can handle that event with JScript. Your code can be a single line that makes Windows Media Player do something simple like play, or it can be a complete application written in JScript.

**See Also**

- [Creating the Skin Definition File](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Adding the Close BUTTONELEMENT**

The Close button is similar in concept to the Play button, but has different codes and colors.

Put the Close **BUTTONELEMENT** code after the closing angle bracket of the Play **BUTTONELEMENT**.

```xml
<BUTTONELEMENT
    mappingColor = "#FF0000"
    upToolTip = "Close"
    onClick="JScript: view.close();" />
```

The following attributes are used to define the **BUTTONELEMENT** for the Close button:

**mappingColor**

This is the color value of the region in the mapping art file you created before. In this case it is the solid red color. This attribute is required for any **BUTTONELEMENT**. By defining this color, you are telling Windows Media Player to associate this color area with the XML code of this button.

**upToolTip**
This defines the text that will be displayed when the user hovers the mouse over the button. This is the same as the Play button except that it is labeled "Close".

**onClick**

This defines the event that occurs when the mouse clicks on the button. The value of this event attribute is called an event handler and will be either a line of Microsoft JScript code, or a JScript function in an external text file that is loaded by the `loadScript` attribute of a VIEW. In this case, the JScript code calls the `close` method of the VIEW element using the global attribute `view`, which closes the view and shuts down Windows Media Player.

**See Also**

- [Creating the Skin Definition File](#)

---

**Complete Code for Simple Skin**

Here is the complete code for the first sample skin:

```html
<THEME>
  <VIEW
    clippingColor = "#CCCC00"
    backgroundImage = "background.bmp"
    titleBar = "false">
    <BUTTONGROUP
      mappingImage = "map.bmp"
      hoverImage = "hover.bmp">
      <BUTTONELEMENT
        mappingColor = "#00FF00"
        upToolTip = "Play"
        onClick="JScript: player.URL='http://proseware.com/laure.wma';" />
      <BUTTONELEMENT
        mappingColor = "#FF0000"
        upToolTip = "Close"
        onClick = "JScript: view.close(); " />
    </BUTTONGROUP>
  </VIEW>
</THEME>
```

Now you have all the pieces put together. Using PKZip, WinZip, or another program that compresses files into
the PKWare "zip" compression format, create a compressed file that contains your skin definition file, bitmaps, and any media files you want to include. Rename the file so that it has a .wmz file name extension. Then double-clicking your compressed skin will start it playing.

You can see a similar working simple skin in the samples section of the SDK.

See Also

- [Creating the Skin Definition File](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

### Adding a Playlist

You can use playlists to choose from collections of your audio and video.

Using the artwork from your first skin, you can make a few changes to the skin definition file.

The first step is to use the shell you will use for most skins:

```xml
<THEME>
  <VIEW
    clippingColor = "#CCCC00"
    backgroundImage = "background.bmp"
    titleBar = "false">
    <BUTTONGROUP
      mappingImage = "map.bmp"
      hoverImage = "hover.bmp">
    </BUTTONGROUP>
  </VIEW>
</THEME>
```

Now add a second **VIEW**, which contains a playlist. Place the following code after the `</VIEW>` of the shell code.

```xml
  <VIEW
    id = "playview">
    <PLAYLIST/>
  </VIEW>
```

You will need to give this second view an ID so you can refer to it later. Add a PLAYELEMENT and a
STOPELEMENT. These predefined buttons make life easier.

```xml
<PLAYELEMENT
    mappingColor = "#00FF00" />
<STOPELEMENT
    mappingColor = "#FF0000" />
```

Finally, add an onClick event to the PLAYELEMENT to display a playlist in the second view:

```xml
onClick = "JScript: theme.openView('playview');" />
```

You can see a similar working playlist skin in the sample section of the SDK.

See Also

- Skin Creation Guide

---

Choosing Files

If you want to choose a file, you can use code similar to the Playlist example, but substitute the following for the PLAYELEMENT section:

```xml
<PLAYELEMENT
    mappingColor = "#00FF00"
    onClick = "JScript:player.URL=theme.openDialog('FILE_OPEN','FILES_ALL');"
/>```

This line will use the openDialog method of THEME to define a URL for the player. You can use this to choose files that are not in playlists.

You can see a similar working openDialog example in the sample section of the SDK.

See Also

- Skin Creation Guide
Adding Video

You can add a video to your file by simply using the \texttt{VIDEO} element and defining where you want the video window to be placed.

Use the same code as you did in the Choosing Files section; all you need to do is add the \texttt{VIDEO} element with the top, left, width, and height attributes.

\begin{verbatim}
<VIDEO
top = "10"
left = "80"
width = "180"
height = "180"/>
\end{verbatim}

Then when you play a video, it will be displayed in the window. The art for the video sample was modified to make a slightly larger skin. Because layers were used in Photoshop, the buttons were easily moved around and a new set was created very quickly. Only the background image was changed. A fill was used in a blank layer and a bevel and emboss effect was added.

You can see a similar working video skin in the sample section of the SDK.

See Also

- \texttt{Skin Creation Guide}
Then you can assign the two buttons a previous and next visualization code string:

```xml
<BUTTONELEMENT
    mappingColor = "#00FF00"
    upToolTip = "Next"
    onClick = "JScript:myeffects.next(); " />
</BUTTONELEMENT

<BUTTONELEMENT
    mappingColor = "#FF0000"
    upToolTip = "Previous"
    onClick = "JScript:myeffects.previous(); " />
</BUTTONELEMENT
```

The layers and bitmaps were the same ones used in the video example, except that the play arrow was copied and flipped horizontally.

Finally, a simple `PLAYER` element with the `URL` attribute was added to choose a song to play.

```xml
<PLAYER
    URL = "http://proseware.com/mellow.wma">
</PLAYER>
```

You can see a similar working visualization skin in the sample section of the SDK.

See Also

- Skin Creation Guide
min = "0"
max = "wmpprop:player.currentMedia.duration"
onmouseup = "player.controls.currentPosition = myslider.value; "
tooltip = "current position"
height = "10"
width = "180"
top = "150"
left = "88"
backgroundColor = "red"
foregroundColor = "blue"
thumbImage = "thumb.bmp"/>

This sets a maximum value based on the duration of the current media file. This uses a tiny thumb image bitmap that is just a 10 pixel by 10 pixel green square. The background of the slider will be red and the foreground will be blue. When the user drags the thumb image to a new position and lets go of the mouse button, the media will change to that position.

But the slider will not move by itself unless you measure the current position with the `currentPosition_onchange` attribute of the `CONTROLS` element, which is embedded in the `PLAYER` element.

<PLAYER
    URL = "http://proseware.com/laure.wma">
    <CONTROLS
        currentPosition_onchange = "myslider.value = player.controls.currentPosition; ">
</PLAYER>

When the position of the media changes, this fires an event which then runs the line of code that changes the value of the slider to the current position of the media.

You can see a similar working slider skin in the sample section of the SDK.

See Also

- Skin Creation Guide

Creating Custom Sliders

You can create custom sliders in any shape you want. For this example, a simple strip is chosen, but the actual shape can be anything. Here is the code for the `CUSTOMSLIDER` element:
<CustomSlider
  top="160"
  left="130"
  min="0"
  max="100"
  toolTip="volume control"
  image="slider.bmp"
  positionImage="graymap.bmp"
  enabled="true"
  value="wmpprop:player.settings.volume"
  value_onchange="player.settings.volume = value" />

This sets up an initial value for the slider. Two new bitmaps are introduced. One is the grayscale bitmap (slider.bmp) that defines which values will be used when clicked on, and the other (slider.bmp) that determines which image will be shown when a particular portion of the grayscale is clicked on.

The initial value is determined by listening to the volume with wmpprop and then the volume can be changed when the user clicks on a portion of the slider that triggers a change in value.

You can see a similar working slider skin in the sample section of the SDK.

See Also

- Skin Creation Guide

Other Skin Samples

To see samples of complete working skins, click the Start button, then point to Programs, then Windows Media, then Windows Media SDK, and then click Player SDK samples. You must first install the Windows Media Player SDK to use the samples.

See Also

- Samples
- Skin Creation Guide
# Skin Programming Reference

The Skin Programming Reference documents the following elements and their associated attributes, methods, and events.

The elements and attributes in this section require Windows Media Player 7.0 or later unless otherwise noted.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient Attributes</td>
<td>Attributes that apply to all skin elements with exceptions noted.</td>
</tr>
<tr>
<td>Ambient Event Handlers</td>
<td>Event handlers that can be implemented by most skin elements.</td>
</tr>
<tr>
<td>Ambient Event Attributes</td>
<td>Attributes detailing the state of Windows Media Player when an event is fired.</td>
</tr>
<tr>
<td>AUTOMENU</td>
<td>Provides a way to display the Quick Access Panel in a skin.</td>
</tr>
<tr>
<td>BUTTON</td>
<td>A standalone button.</td>
</tr>
<tr>
<td>BUTTONELEMENT</td>
<td>A button within a button group.</td>
</tr>
<tr>
<td>BUTTONGROUP</td>
<td>A group of button elements.</td>
</tr>
<tr>
<td>COLUMN</td>
<td>Represents a column within a playlist control.</td>
</tr>
<tr>
<td>CONTROLS</td>
<td>Provides access to the Controls object from within a skin.</td>
</tr>
<tr>
<td>CUSTOMSLIDER</td>
<td>A customizable slider control.</td>
</tr>
<tr>
<td>EDITBOX</td>
<td>Provides a way for users to enter text within a skin.</td>
</tr>
<tr>
<td>EFFECTS</td>
<td>An element that contains and controls a collection of effects.</td>
</tr>
<tr>
<td>EQUALIZERSETTINGS</td>
<td>An element allowing manipulation of the graphic equalizer.</td>
</tr>
<tr>
<td>ITEM</td>
<td>Represents an item in a list box or pop-up control.</td>
</tr>
<tr>
<td>LISTBOX</td>
<td>Provides a way for users to select items from a list.</td>
</tr>
<tr>
<td>PLAYER</td>
<td>Provides access to the Player object from within a skin.</td>
</tr>
<tr>
<td>PLAYLIST</td>
<td>An element for controlling the appearance of a playlist within a skin.</td>
</tr>
<tr>
<td>POPUP</td>
<td>Provides a way for users to select items from a list.</td>
</tr>
<tr>
<td>PROGRESSBAR</td>
<td>Provides a way to display progress information in a horizontal or vertical control.</td>
</tr>
<tr>
<td>SETTINGS</td>
<td>Provides access to the Settings object from within a skin.</td>
</tr>
</tbody>
</table>
Ambient Attributes

The ambient attributes and method are supported by all appropriate skin elements, except where noted.

The following attributes are ambient.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accName</td>
<td>Specifies or retrieves a name for any element.</td>
</tr>
<tr>
<td>accDescription</td>
<td>Specifies or retrieves a description for any element.</td>
</tr>
<tr>
<td>accKeyboardShortcut</td>
<td>Specifies or retrieves a keyboard shortcut description for any element.</td>
</tr>
<tr>
<td>alphaBlend</td>
<td>Specifies or retrieves a value for alpha blending any VIEW, SUBVIEW, or UI widget.</td>
</tr>
<tr>
<td>clippingColor</td>
<td>Specifies or retrieves the color to clip out from the clippingImage bitmap.</td>
</tr>
<tr>
<td>clippingImage</td>
<td>Specifies or retrieves the region to clip the control to.</td>
</tr>
<tr>
<td>elementType</td>
<td>Retrieves the type of the element (for instance, BUTTON).</td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies or retrieves a value indicating whether the control is enabled or disabled.</td>
</tr>
</tbody>
</table>
**height**

Specifies or retrieves the height of the control.

**horizontalAlignment**

Specifies or retrieves a value that indicates the horizontal placement of the control when the **VIEW** or parent **SUBVIEW** is resized.

**id**

Specifies or retrieves the identifier of a control. Can only be set at design time.

**left**

Specifies or retrieves the left coordinate of the control.

**passThrough**

Specifies or retrieves a value indicating whether the control will pass all mouse events through to the control under it.

**tabStop**

Specifies or retrieves a value indicating whether the control will be in the tabbing order.

**top**

Specifies or retrieves the top coordinate of the control.

**verticalAlignment**

Specifies or retrieves a value that indicates the vertical placement of the control when the **VIEW** or parent **SUBVIEW** is resized.

**visible**

Specifies or retrieves the visibility of the control.

**width**

Specifies or retrieves the width of the control.

**zIndex**

Specifies or retrieves the order in which the control is rendered.

The following methods are ambient.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>alphaBlendTo</strong></td>
<td>Adjusts the <strong>alphaBlend</strong> property over a period of time.</td>
</tr>
<tr>
<td><strong>moveTo</strong></td>
<td>Moves the control.</td>
</tr>
</tbody>
</table>

**See Also**

- Skin Programming Reference

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**AmbientAttributes.accName**

The **accName** attribute specifies or retrieves a name for any element.
**Syntax**

*elementID*.accName

**Possible Values**

This attribute is a read/write String with a default value equal to the id attribute.

**Remarks**

This attribute is used for accessibility purposes. It allows the name of any element to be read aloud by a reader program.

This attribute also applies to button elements inside a button group control.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- Ambient Attributes

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**AmbientAttributes.accDescription**

The accDescription attribute specifies or retrieves a description for any element.

**Syntax**

*elementID*.accDescription

**Possible Values**

This attribute is a read/write String with a default value of "" (empty string).

**Remarks**

This attribute is used for accessibility purposes. It allows the description of any element to be read aloud by a reader program.
This attribute also applies to button elements inside the button group control.

Requirements

Windows Media Player 9 Series or later.

See Also

- Ambient Attributes

AmbientAttributes.accKeyboardShortcut

The *accKeyboardShortcut* attribute specifies or retrieves a keyboard shortcut description for any element.

Syntax

```
elementID.accKeyboardShortcut
```

Possible Values

This attribute is a read/write *String* with a default value of "" (empty string) For the *BUTTON* element, this attribute has a default value of "Spacebar or Enter". For the *SLIDER* element, the default value is "Right/Up Arrow to increase, Left/Down Arrow to decrease".

Remarks

This attribute is used for accessibility purposes. It allows a description of the keyboard shortcut for any element to be read aloud by a reader program. This attribute does not set the shortcut. The scripter must do that work.

This attribute also applies to button elements inside the button group control.

Requirements

Windows Media Player 9 Series or later.

See Also

- Ambient Attributes
The `alphaBlend` attribute specifies or retrieves a value for alpha blending any `VIEW`, `SUBVIEW`, or UI widget.

**Syntax**

```
elementID.alphaBlend
```

**Possible Values**

This attribute is a read/write `Number (long)` with a value ranging from 0 (no opacity) to 255 (full opacity) and a default value of 255.

**Remarks**

This attribute allows an element to appear semitransparent, depending on the amount of opacity set. The less opacity, the more transparent the element will appear. Each element in the skin can have a separate opacity value except for button elements in a `BUTTONGROUP` control. When `alphaBlend` is set in `VIEW`, the opacity of the entire skin will be set. Alpha blend will not work for windowed controls, including `PLAYLIST`, `EFFECTS`, `LISTBOX`, `POPUP`, `EDITBOX`, and `VIDEO` (if `windowless` is set to false). When `alphaBlend` is set on `VIEW`, the whole skin becomes transparent. The `transparencyColor` attributes used by several elements are not supported with `alphaBlend`.

When you use `alphaBlend` with a `TEXT` element that does not have the `backgroundColor` specified, a background color of black will be used. If the foreground color is also black (which is the default value for `TEXT.foregroundColor`), your text may become unreadable. In order to prevent this, always specify the `backgroundColor` attribute, or set `foregroundColor` to a color other than black.

**Note**  This attribute is not supported in Windows 98.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [Ambient Attributes](#)
- [AmbientAttributes.alphaBlendTo](#)
- [TEXT Element](#)
AmbientAttributes.alphaBlendTo

The `alphaBlendTo` method adjusts the `alphaBlend` property over a period of time.

**Syntax**

```
functionName = elementID.alphaBlendTo(newVal, alphaTime)
```

**Parameters**

- `newVal`  
  Number (`long`) specifying the new opacity value. Ranges from 0 (no opacity) to 255 (full opacity).

- `alphaTime`  
  Number (`long`) specifying the time, in milliseconds, that it takes the element to change opacity.

**Return Values**

This method does not return a value.

**Remarks**

This method is useful for making elements gradually appear or disappear.

When you use `alphaBlendTo` with a `TEXT` element that does not have the `backgroundColor` specified, a background color of black will be used. If the foreground color is also black (which is the default value for `TEXT.foregroundColor`), your text may become unreadable. In order to prevent this, always specify the `backgroundColor` attribute, or set `foregroundColor` to a color other than black.

**Note**  
This attribute is not supported in Windows 98.

**Requirements**

Windows Media Player 9 Series or later.
See Also

- Ambient Attributes
- AmbientAttributes.alphaBlend
- TEXT Element
- TEXT.backgroundColor
- TEXT.foregroundColor

**AmbientAttributes.clippingColor**

The **clippingColor** attribute specifies or retrieves the color to clip out from the **clippingImage** bitmap.

**Syntax**

```
elementID.clippingColor
```

**Possible Values**

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>Default. The color at pixel location 0,0 is used.</td>
</tr>
<tr>
<td><em>any Microsoft Internet Explorer color value</em></td>
<td>The specified Internet Explorer color is used.</td>
</tr>
</tbody>
</table>

**Remarks**

The clipping color indicates the regions of the **clippingImage** (or **backgroundImage** for VIEW or SUBVIEW) that correspond to transparent, non-clickable portions of the control. The clipping color can indicate multiple regions to be clipped out. A warning is issued if the **clippingImage** is a JPG to warn of loss of color in JPGs.

The **clippingColor** attribute is not supported by the **PLAYLIST** element.

**Requirements**

Windows Media Player version 7.0 or later.
See Also

- Ambient Attributes
- AmbientAttributes.clippingImage
- Color Reference

AmbientAttributes.clippingImage

The **clippingImage** attribute specifies or retrieves the region to clip the control to.

**Syntax**

```plaintext
elementID.clippingImage
```

**Possible Values**

This attribute is a read/write **String** indicating the image file name. It has no default value.

**Remarks**

The **clippingImage** attribute supports PNG, JPG, BMP, and GIF files (not including animated GIFs). Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended for clipping images.

This attribute is useful when you want to display only a part of the control image and not the entire rectangular area. The **clippingColor** attribute indicates the regions of the clipping image that correspond to transparent, non-clickable portions of the control. The control can therefore be of any shape. For best results, the clipping image should be the same size as the control image.

The **clippingImage** attribute is not supported by the **PLAYLIST**, **VIEW**, and **SUBVIEW** elements. A clipping image will not work with the **VIDEO** element if **VIDEO.windowless** is set to false, nor with the **EFFECTS** element if **EFFECTS.windowed** is set to true.

Because the use of clipping images imposes a performance penalty, they should not be used when efficiency is an issue.

**Example**

See the **BUTTONELEMENT.mappingColor** attribute for a sample illustrating the use of this attribute.
Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
- AmbientAttributes.clippingColor

AmbientAttributes.elementType

The `elementType` attribute retrieves the type of the element (for instance, BUTTON).

Syntax

```
elementID.elementType
```

Possible Values

This attribute is a read-only `String` indicating the name of the element.

Remarks

This attribute is useful in determining the type of element that fired an event and writing a generic event handler for this class of elements.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
AmbientAttributes.enabled

The enabled attribute specifies or retrieves a value indicating whether the control is enabled or disabled.

Syntax

elementID.enabled

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Control enabled.</td>
</tr>
<tr>
<td>false</td>
<td>Control disabled.</td>
</tr>
</tbody>
</table>

Remarks

If the control is enabled, it can have a tab stop, and will receive all ambient events. When disabled, the control does not have a tab stop, and does not receive any ambient mouse or keyboard events fired to it. (However, it will continue to receive all other ambient events fired to it.)

This attribute is not supported for the SUBVIEW element.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
AmbientAttributes.height

The **height** attribute specifies or retrieves the height of the control.

**Syntax**

`elementID.height`

**Possible Values**

This attribute is a read/write **Number (long)** representing the height of the control in pixels. The default value is zero or the height of the image specified in the control's **image** attribute.

**Remarks**

If the height specified is smaller than the height of the image provided, then the image will be clipped. If the height is larger than the height of the image, then the click region will go beyond the image boundary. No matter what value is given to this attribute, the image cannot grow beyond its parent **VIEW** or **SUBVIEW**.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes

---

AmbientAttributes.horizontalAlignment

The **horizontalAlignment** attribute specifies or retrieves a value that indicates the horizontal placement of the control when the **VIEW** or parent **SUBVIEW** is resized.

**Syntax**

`elementID.horizontalAlignment`
Possible Values

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Default. Maintains the placement of the control relative to the left of the VIEW or parent SUBVIEW when the view is resized.</td>
</tr>
<tr>
<td>right</td>
<td>Maintains the placement of the control relative to the right of the VIEW or parent SUBVIEW when the view is resized.</td>
</tr>
<tr>
<td>center</td>
<td>Maintains the placement of the control relative to the horizontal center of the VIEW or parent SUBVIEW when the view is resized.</td>
</tr>
<tr>
<td>stretch</td>
<td>Maintains the placement of the control relative to both the left and right margins of the VIEW or parent SUBVIEW when resized. The control stretches to fit when the VIEW or SUBVIEW is stretched. The actual image does not grow or shrink unless it is resizable, but the clickable area grows or shrinks if not bounded by a clippingImage.</td>
</tr>
</tbody>
</table>

Remarks

Unless **horizontalAlignment** is set to "center", the control retains its original distance from the specified edge, or from both edges if "stretch" is specified and the control is resizable. If the control is not resizable and "stretch" is specified, the clickable region is stretched instead.

You can set any combination of **horizontalAlignment** and **verticalAlignment**. For example, if you want to center a control both horizontally and vertically, set horizontalAlignment="center" verticalAlignment="center".

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
- AmbientAttributes.verticalAlignment
- AmbientAttributes.clippingImage

© 2000-2003 Microsoft Corporation. All rights reserved.
**AmbientAttributes.id**

The **id** attribute specifies or retrieves the identifier of a control. This attribute will support any name permitted by Microsoft Visual Basic for Applications. Can only be set at design time.

**Syntax**

`elementID.id`

**Possible Values**

This attribute is a **String** specified at design time and read-only thereafter. It has a default value of

`Unnamed_elementtype_num`

where `elementtype` is the name of the control type, and `num` is a number corresponding to the position of the control in the sequence of all unnamed controls in the Theme. This number is reset to zero for each new Theme.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes

---

**AmbientAttributes.left**

The **left** attribute specifies or retrieves the left coordinate of the control.

**Syntax**

`elementID.left`

**Possible Values**

This attribute is a read/write **Number (long)** representing the distance in pixels from the control to the left edge of the parent **VIEW** or **SUBVIEW**. It has a default value of zero. Negative numbers are allowed, in which case the left border of the **VIEW** or **SUBVIEW** clips the control.
Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes

AmbientAttributes.moveTo

The \texttt{moveTo} method moves the control.

Syntax

\begin{verbatim}
    elementID.moveTo(newLeft, newTop, time)
\end{verbatim}

Parameters

\begin{itemize}
    \item \texttt{newX} \texttt{Number (long)} specifying the new value for the \texttt{left} attribute of the control.
    \item \texttt{newY} \texttt{Number (long)} specifying the new value for the \texttt{top} attribute of the control.
    \item \texttt{time} \texttt{Number (long)} specifying the time, in milliseconds, that it takes for the control to move to its new location.
\end{itemize}

Return Values

This method does not return a value.

Remarks

This method is useful for animated \texttt{SUBVIEW} elements (for example, if the user clicks on a tray and the controls slide down).
Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
- AmbientAttributes.left
- AmbientAttributes.top

AmbientAttributes.passThrough

The passThrough attribute specifies or retrieves a value indicating whether the control will pass all mouse events through to the control under it.

Syntax

```xml
elementID.passThrough
```

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Control passes events through to the control under it.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Control does not pass events through.</td>
</tr>
</tbody>
</table>

Remarks

This attribute is useful if, for example, a text control sits on top of a button control only to provide labeling. In this case, clicks on the text control must be passed through to the button control.

The passThrough attribute is not supported by the VIEW, SUBVIEW, and PLAYLIST elements. It will not work with the VIDEO element if VIDEO.windowless is set to false, nor with the EFFECTS element if EFFECTS.windowed is set to true.

Requirements
AmbientAttributes.tabStop

The `tabStop` attribute specifies or retrieves a value indicating whether the control is in the tabbing order. You set the tabbing order by placing the control in the overall script before or after other control tags.

**Syntax**

```
elementID.tabStop
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Control is in the tabbing order (control will have a tab stop: accessibility requirement).</td>
</tr>
<tr>
<td>false</td>
<td>Control is not in the tabbing order.</td>
</tr>
</tbody>
</table>

**Remarks**

The `tabStop` attribute is not applicable to the `EFFECTS` element.

The default value for this attribute is true for all elements except `AUTOMENU` and `TEXT`, which have a default value of false.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes
AmbientAttributes.top

The **top** attribute specifies or retrieves the top coordinate of the control.

**Syntax**

```
elementID.top
```

**Possible Values**

This attribute is a read/write **Number (long)** representing the distance in pixels from the control to the top edge of the parent **VIEW** or **SUBVIEW**. It has a default value of zero. Negative numbers are allowed, in which case the top border of the **VIEW** or **SUBVIEW** clips the control.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes
- AmbientAttributes.id

AmbientAttributes.verticalAlignment
The `verticalAlignment` attribute specifies or retrieves a value indicating the vertical placement of the control when the `VIEW` or parent `SUBVIEW` is stretched.

**Syntax**

```
elementID.verticalAlignment
```

**Possible Values**

This attribute is a read/write `String`.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>Default. Maintains the placement of the control relative to the top of the <code>VIEW</code> or parent <code>SUBVIEW</code> when the view is resized.</td>
</tr>
<tr>
<td>bottom</td>
<td>Maintains the placement of the control relative to the bottom of the <code>VIEW</code> or parent <code>SUBVIEW</code> when the view is resized.</td>
</tr>
<tr>
<td>center</td>
<td>Maintains the placement of the control relative to the vertical center of the <code>VIEW</code> or parent <code>SUBVIEW</code> when the view is resized.</td>
</tr>
<tr>
<td>stretch</td>
<td>Maintains the placement of the control relative to both the top and bottom margins of the View when resized. The control stretches to fit when the <code>VIEW</code> or parent <code>SUBVIEW</code> is stretched. The actual image does not grow or shrink unless it is resizable, but the clickable area grows or shrinks if not bounded by a <code>clippingImage</code>.</td>
</tr>
</tbody>
</table>

**Remarks**

Unless `verticalAlignment` is set to "center", the control retains its original distance from the specified edge, or from both edges if "stretch" is specified and the control is resizable. If the control is not resizable and "stretch" is specified, the clickable region is stretched instead.

You can set any combination of the `horizontalAlignment` and `verticalAlignment` attributes. For example, if you want a control to be centered both horizontally and vertically, set `horizontalAlignment` to "center" and `verticalAlignment` to "center".

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes
- AmbientAttributes.horizontalAlignment

© 2000-2003 Microsoft Corporation. All rights reserved.
AmbientAttributes.visible

The visible attribute specifies or retrieves the visibility for the control.

Syntax

elementID.visible

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The control is visible.</td>
</tr>
<tr>
<td>false</td>
<td>The control is not visible.</td>
</tr>
</tbody>
</table>

Remarks

This attribute is useful for hiding controls, for example when swapping a pause button for a play button.

If the value is false, the control is not visible and click events are passed to the control behind it. If the value is true, the control is visible and receives the click event itself.

The default value for the AUTOMENU element is false.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes
AmbientAttributes.width

The **width** attribute specifies or retrieves the width of the control.

**Syntax**

```
elementID.width
```

**Possible Values**

This attribute is a read/write 16-bit **Integer** (0 to 32,767) representing the width of the control in pixels. The default value is zero or the width of the image specified in the control's **image** attribute.

**Remarks**

If the width specified is narrower than the width of the image provided, then the image will be clipped. If the width is wider than the width of the image, then the click region will go beyond the image boundary. No matter what value is given to this attribute, the image cannot grow beyond its parent **VIEW** or **SUBVIEW**.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Attributes

---

AmbientAttributes.zIndex

The **zIndex** attribute specifies or retrieves the order in which the control is rendered.

**Syntax**

```
elementID.zIndex
```

**Possible Values**
This attribute is a read/write Number (long) with a default value of zero. The range is that of a signed long integer.

Remarks

The background bitmap of a VIEW or SUBVIEW has a fixed z index of zero. If you want a control to be behind the background, the zIndex must be set to a negative number.

The z index of a VIEW or SUBVIEW is an absolute index, while the z index of a control is relative to the z index of the VIEW or SUBVIEW that contains it.

The zIndex attribute is not supported by the BROWSER and PLAYLIST elements. It will not work with the VIDEO element if VIDEO.windowless is set to false, nor with the EFFECTS element if EFFECTS.windowed is set to true.

BUTTONELEMENT elements use the zIndex of their BUTTONGROUP.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Attributes

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_onchange</td>
<td>When a skin attribute changes value, an event occurs that can be handled by an event handler. The name of the event handler is the name of the attribute followed by an underscore and &quot;onchange&quot;, such as value_onchange.</td>
</tr>
<tr>
<td>onblur</td>
<td>Handles an event that occurs when the element loses the keyboard focus.</td>
</tr>
</tbody>
</table>
attribute_onchange

When a skin attribute changes value, an event occurs that can be handled by an event handler. The name of the event handler is the name of the attribute followed by an underscore and "onchange", such as `value_onchange`.
Syntax

_attribute_onchange_

Example

<SLIDER
    thumbImage = "thumb.gif"
    value_onchange = "JScript: if (value == 100) backgroundColor = 'green';"
/>

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Handlers

onblur

The onblur event handler handles an event that occurs when the element loses the keyboard focus.

Syntax

onblur

Remarks

The onblur event handler is not applicable to the EFFECTS element or the POPUP element.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Handlers
The `onclick` event handler handles an event that occurs when the user clicks the element.

**Syntax**

```javascript
onclick
```

**Remarks**

This event handler is not applicable to the `PLAYLIST` element, the `POPUP` element, the `VIDEO` element, the `LISTBOX` element, when `VIDEO.windowless` is false, or the `EFFECT` element when `EFFECTS.windowed` is true.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers

---

The `ondblclick` event handler handles an event that occurs when the user double-clicks the element.

**Syntax**

```javascript
ondblclick
```
ondblclick

**Remarks**

This event handler is not applicable to the `PLAYLIST` element, the `POPUP` element, the `VIDEO` element when `VIDEO.windowless` is false, or the `EFFECT` element when `EFFECTS.windowed` is true.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Ambient Event Handlers](#)

---

onendalphablend

The `onendalphablend` event handler handles an event that occurs when an element completes an `alphaBlendTo` operation.

**Syntax**

`onendalphablend`

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [Ambient Event Handlers](#)
- [AmbientAttributes.alphaBlendTo](#)
onendmove

The onendmove event handler handles an event that occurs when an element completes a moveTo operation.

Syntax

onendmove

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Handlers
- AmbientAttributes.moveTo

onfocus

The onfocus event handler handles an event that occurs when the element receives the keyboard focus.

Syntax

onfocus

Remarks

The onfocus event handler is not applicable to the EFFECTS element or the POPUP element.

Requirements

Windows Media Player version 7.0 or later.
**onkeydown**

The **onkeydown** event handler handles an event that occurs when a key is pressed.

**Syntax**

```
onkeydown
```

**Remarks**

This event handler is not applicable to the **PLAYLIST** element, the **EFFECTS** element, or the **VIDEO** element when `VIDEO.windowless` is false.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers

---

**onkeypress**

---

See Also

- Ambient Event Handlers
The `onkeypress` event handler handles an event that occurs when an alphanumerical key is pressed.

**Syntax**

```javascript
onkeypress
```

**Remarks**

This event handler is not applicable to the `PLAYLIST` element, the `EFFECTS` element, or the `VIDEO` element when `VIDEO.windowless` is false.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers

---

The `onkeyup` event handler handles an event that occurs when a key is released.

**Syntax**

```javascript
onkeyup
```

**Remarks**

This event handler is not applicable to the `PLAYLIST` element, the `EFFECTS` element, or the `VIDEO` element when `VIDEO.windowless` is false.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers
onmousedown

The `onmousedown` event handler handles an event that occurs when the user clicks a mouse button.

**Syntax**

`onmousedown`

**Remarks**

This event handler is not applicable to the `PLAYLIST` element, the `VIDEO` element when `VIDEO.windowless` is false, or the `EFFECT` element when `EFFECTS.windowed` is true.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Ambient Event Handlers](#)

onmousemove

The `onmousemove` event handler handles an event that occurs when the user moves the mouse pointer while it is over an element.

**Syntax**

-
onmousemove

Remarks
This event handler is not applicable to the PLAYLIST element, the VIDEO element when VIDEO.windowless is false, or the EFFECT element when EFFECTS.windowed is true.

Requirements
Windows Media Player version 7.0 or later.

See Also

• Ambient Event Handlers

onmouseout

The onmouseout event handler handles an event that occurs when the user moves the pointer off the element.

Syntax

onmouseout

Remarks
This event handler is not applicable to the PLAYLIST element.

Requirements
Windows Media Player version 7.0 or later.

See Also

• Ambient Event Handlers
onmouseover

The onmouseover event handler handles an event that occurs when the user first places the pointer over the element.

Syntax

```javascript
onmouseover
```

Remarks

This event handler is not applicable to the PLAYLIST element.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Handlers

onmouseup

The onmouseup event handler handles an event that occurs when the user releases a mouse button while the pointer is over the element.

Syntax

```javascript
onmouseup
```

Remarks
This event handler is not applicable to the **PLAYLIST** element.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers

---

**onresize**

The **onresize** event handler handles an event that occurs when a control resizes.

**Syntax**

```plaintext
onresize
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Handlers

---

© 2000-2003 Microsoft Corporation. All rights reserved.
The following attributes are common to all skin events. They are accessed with the `event` keyword within an event handler for the element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>altKey</code></td>
<td>Retrieves a value indicating whether the ALT key was down when the event occurred.</td>
</tr>
<tr>
<td><code>button</code></td>
<td>Retrieves a value indicating which mouse buttons were down when the event occurred.</td>
</tr>
<tr>
<td><code>clientX</code></td>
<td>Retrieves the x coordinate of the mouse pointer with respect to the client region of the application window.</td>
</tr>
<tr>
<td><code>clientY</code></td>
<td>Retrieves the y coordinate of the mouse pointer with respect to the client region of the application window.</td>
</tr>
<tr>
<td><code>ctrlKey</code></td>
<td>Retrieves a value indicating whether the CTRL key was down when the event occurred.</td>
</tr>
<tr>
<td><code>fromElement</code></td>
<td>Retrieves the element the event came from.</td>
</tr>
<tr>
<td><code>keyCode</code></td>
<td>Retrieves the ASCII key code if a key was pressed when the event occurred.</td>
</tr>
<tr>
<td><code>offsetX</code></td>
<td>Retrieves the x coordinate of the mouse pointer with respect to the element firing the event.</td>
</tr>
<tr>
<td><code>offsetY</code></td>
<td>Retrieves the y coordinate of the mouse pointer with respect to the element firing the event.</td>
</tr>
<tr>
<td><code>screenHeight</code></td>
<td>Retrieves the height of the available screen size in pixels.</td>
</tr>
<tr>
<td><code>screenWidth</code></td>
<td>Retrieves the width of the available screen size in pixels.</td>
</tr>
<tr>
<td><code>screenX</code></td>
<td>Retrieves the absolute x coordinate of the mouse pointer with respect to the screen.</td>
</tr>
<tr>
<td><code>screenY</code></td>
<td>Retrieves the absolute y coordinate of the mouse pointer with respect to the screen.</td>
</tr>
<tr>
<td><code>shiftKey</code></td>
<td>Retrieves a value indicating whether the SHIFT key was down when the event occurred.</td>
</tr>
<tr>
<td><code>srcElement</code></td>
<td>Retrieves the element that fired the event.</td>
</tr>
<tr>
<td><code>toElement</code></td>
<td>Retrieves the element the keyboard focus moved to.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Retrieves the x coordinate of the mouse pointer with respect to the application window.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Retrieves the y coordinate of the mouse pointer with respect to the application window.</td>
</tr>
</tbody>
</table>

See Also

- Skin Programming Reference
event.altKey

The `altKey` attribute retrieves a value indicating whether the ALT key was down when the event occurred.

Syntax

```
event.altKey
```

Possible Values

This attribute is a read-only `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Indicates the ALT key was in the down position.</td>
</tr>
<tr>
<td>false</td>
<td>Indicates the ALT key was in the up position.</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes

---

**event.button**
The **button** attribute retrieves a value indicating which mouse buttons were down when the event occurred.

**Syntax**

`event.button`

**Possible Values**

This attribute is a read-only **Number (long)**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No mouse buttons were down.</td>
</tr>
<tr>
<td>1</td>
<td>The left mouse button was down.</td>
</tr>
<tr>
<td>2</td>
<td>The right mouse button was down.</td>
</tr>
<tr>
<td>3</td>
<td>Both mouse buttons were down.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Ambient Event Attributes](#)

---

**event.clientX**

The **clientX** attribute retrieves the x coordinate of the mouse pointer with respect to the client region of the application window.

**Syntax**

`event.clientX`

**Possible Values**

This attribute is a read-only **Number (long)**.
Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes
- `event.clientX`

`event.clientY`

The `cliency` attribute retrieves the y coordinate of the mouse pointer with respect to the client region of the application window.

Syntax

`event.clientY`

Possible Values

This attribute is a read-only **Number (long)**.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes
- `event.clientX`
**event.ctrlKey**

The `ctrlKey` attribute retrieves a value indicating whether the CTRL key was down when the event occurred.

**Syntax**

`event.ctrlKey`

**Possible Values**

This attribute is a read-only `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Indicates the CTRL key was in the down position.</td>
</tr>
<tr>
<td>false</td>
<td>Indicates the CTRL key was in the up position.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Attributes

---

**event.fromElement**

The `fromElement` attribute retrieves the element the event came from. If no element passes an event, the value is `NULL`.

**Syntax**
event.fromElement

Possible Values

This attribute is a read-only object.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes

---

event.keyCode

The keyCode attribute retrieves the ASCII key code if a key was pressed when the event occurred.

Syntax

event.keyCode

Possible Values

This attribute is a read-only Number (long).

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes

---

© 2000-2003 Microsoft Corporation. All rights reserved.
event.offsetX

The `offsetX` attribute retrieves the x coordinate of the mouse pointer with respect to the element firing the event.

Syntax

```javascript
event.offsetX
```

Possible Values

This attribute is a read-only `Number (long)`.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes
- `event.offsetY`

---

event.offsetY

The `offsetY` attribute retrieves the y coordinate of the mouse pointer with respect to the element firing the event.

Syntax

```javascript
event.offsetY
```

Possible Values

This attribute is a read-only `Number (long)`.
event.screenHeight

The `screenHeight` attribute retrieves the height of the available screen size in pixels.

Syntax

event.screenHeight

Possible Values

This attribute is a read-only `Number (long)`.

Remarks

This is useful for determining the amount of space there is in the monitor. If there are two monitors, it calculates the space for both monitors.

Requirements

Windows Media Player 9 Series or later.

See Also

- Ambient Event Attributes
- `event.offsetX`
- `event.screenWidth`
event.screenWidth

The `screenWidth` attribute retrieves the width of the available screen size in pixels.

Syntax

event.screenWidth

Possible Values

This attribute is a read-only `Number (long)`.

Remarks

This is useful for determining the amount of space there is in the monitor. If there are two monitors, it calculates the space for both monitors.

Requirements

Windows Media Player 9 Series or later.

See Also

- Ambient Event Attributes
- `event.screenHeight`

event.screenX

The `screenX` attribute retrieves the absolute x coordinate of the mouse pointer with respect to the screen.

Syntax
**event.screenX**

**Possible Values**

This attribute is a read-only `Number (long)`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Attributes
- `event.screenY`

---

**event.screenY**

The `screenY` attribute retrieves the absolute y coordinate of the mouse pointer with respect to the screen.

**Syntax**

`event.screenY`

**Possible Values**

This attribute is a read-only `Number (long)`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Attributes
- `event.screenX`
**event.shiftKey**

The **shiftKey** attribute retrieves a value indicating whether the SHIFT key was down when the event occurred.

**Syntax**

```
event.shiftKey
```

**Possible Values**

This attribute is a read-only **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Indicates the SHIFT key was in the down position.</td>
</tr>
<tr>
<td>false</td>
<td>Indicates the SHIFT key was in the up position.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Ambient Event Attributes](#)

---

**event.srcElement**

The **srcElement** attribute retrieves the element that fired the event.

**Syntax**

```
```

---

© 2000-2003 Microsoft Corporation. All rights reserved.
**event.srcElement**

**Possible Values**

This attribute is a read-only object.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Attributes

---

**event.toElement**

The `toElement` attribute retrieves the element that the keyboard focus moved to. This attribute only applies to the `onblur` event; for all other events, its value is `NULL`.

**Syntax**

`event.toElement`

**Possible Values**

This attribute is a read-only object.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Ambient Event Attributes

© 2000-2003 Microsoft Corporation. All rights reserved.
**event.x**

The `x` attribute retrieves the x coordinate of the mouse pointer with respect to the application window.

**Syntax**

```
event.x
```

**Possible Values**

This attribute is a read-only `Number (long)`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Ambient Event Attributes](#)
- `event.y`

---

**event.y**

The `y` attribute retrieves the y coordinate of the mouse pointer with respect to the application window.

**Syntax**

```
event.y
```

**Possible Values**

This attribute is a read-only `Number (long)`.

---

© 2000-2003 Microsoft Corporation. All rights reserved.
Requirements

Windows Media Player version 7.0 or later.

See Also

- Ambient Event Attributes
- event.x

AUTOMENU Element

The AUTOMENU element provides a way to display the Quick Access Panel in a skin. This is a menu that appears in the full mode of the Player when you click the arrow to the right of the Now Playing button. It provides immediate access to digital media files organized in various ways, such as in playlists or by album, artist, or genre. Use the left and top attributes to specify the location at which the menu should appear when the show method is called.

The AUTOMENU element supports the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td>Displays the Quick Access Panel.</td>
</tr>
</tbody>
</table>

The AUTOMENU element supports the following ambient attributes: left and top.

Note   This element requires Windows Media Player 9 Series or later.

See Also

- Skin Programming Reference
AUTOMENU.show

The show method displays the Quick Access Panel.

Syntax

elementID.show("Play")

Parameters

"Play"

Always specify this value.

Return Values

This method does not return a value.

Remarks

The Quick Access Panel appears at the location specified by the left and top attributes of the AUTOMENU element.

Requirements

Windows Media Player 9 Series or later.

See Also

- AUTOMENU Element

© 2000-2003 Microsoft Corporation. All rights reserved.
The **BUTTON** element provides a way to create a button within a skin. The attributes below can be used to customize the behavior of a button, or a predefined button can be used for convenience.

The **BUTTON** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cursor</strong></td>
<td>Specifies or retrieves the cursor that appears when the mouse pointer hovers over the <strong>BUTTON</strong>.</td>
</tr>
<tr>
<td><strong>disabledImage</strong></td>
<td>Specifies or retrieves the image that displays when the <strong>BUTTON</strong> is disabled.</td>
</tr>
<tr>
<td><strong>down</strong></td>
<td>Specifies or retrieves a value indicating whether the <strong>BUTTON</strong> is in the up or down position.</td>
</tr>
<tr>
<td><strong>downImage</strong></td>
<td>Specifies or retrieves the image representing the down state of the <strong>BUTTON</strong>.</td>
</tr>
<tr>
<td><strong>downToolTip</strong></td>
<td>Specifies or retrieves the ToolTip text that appears when the mouse is over the <strong>BUTTON</strong> and the <strong>BUTTON</strong> is in the down or depressed state.</td>
</tr>
<tr>
<td><strong>hoverDownImage</strong></td>
<td>Specifies or retrieves the image displayed when the <strong>BUTTON</strong> is in the down state and the user hovers over it with the mouse pointer.</td>
</tr>
<tr>
<td><strong>hoverImage</strong></td>
<td>Specifies or retrieves the image displayed when the <strong>BUTTON</strong> is in the up state and the user hovers over it with the mouse pointer.</td>
</tr>
<tr>
<td><strong>image</strong></td>
<td>Specifies or retrieves the default image of the <strong>BUTTON</strong>.</td>
</tr>
<tr>
<td><strong>sticky</strong></td>
<td>Specifies or retrieves a value indicating whether the <strong>BUTTON</strong> is a toggle, that is, whether it is a two-state or single-state button.</td>
</tr>
<tr>
<td><strong>tiled</strong></td>
<td>Specifies or retrieves a value indicating whether the <strong>BUTTON</strong> image will be tiled.</td>
</tr>
<tr>
<td><strong>transparencyColor</strong></td>
<td>Specifies or retrieves the color that will be transparent in the <strong>BUTTON</strong> image.</td>
</tr>
<tr>
<td><strong>upToolTip</strong></td>
<td>Specifies or retrieves the ToolTip text that appears when the mouse is over the <strong>BUTTON</strong> and the <strong>BUTTON</strong> is in the up state.</td>
</tr>
</tbody>
</table>

The **BUTTON** element supports the ambient attributes and can implement the ambient event handlers. For more information, see [Ambient Attributes](#) and [Ambient Event Handlers](#).

Predefined buttons are normal **BUTTON** elements with various common attribute settings specified by default. The following predefined buttons are available.

<table>
<thead>
<tr>
<th>Predefined <strong>BUTTON</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CLOSEBUTTON</strong></td>
<td>A <strong>BUTTON</strong> used to close the Player.</td>
</tr>
<tr>
<td><strong>FFWDBUTTON</strong></td>
<td>A <strong>BUTTON</strong> with a built in call to <code>player.controls.fastForward</code> when clicked.</td>
</tr>
<tr>
<td><strong>IMAGEBUTTON</strong></td>
<td>A <strong>BUTTON</strong> used to display an image.</td>
</tr>
</tbody>
</table>
BUTTON.cursor

The cursor attribute specifies or retrieves the cursor that appears when the mouse pointer hovers over the BUTTON.

Syntax

elementID.cursor

Possible Values
This attribute is a read/write String.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Default. Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file)</td>
</tr>
</tbody>
</table>

Remarks

If the system does not recognize the cursor value specified, the cursor value remains at the previously set value.

Cursor file name paths are ignored, so the cursor file must reside in the default directory.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element

© 2000-2003 Microsoft Corporation. All rights reserved.

BUTTON.disabledImage

The disabledImage attribute specifies or retrieves the image that displays when the BUTTON is disabled.
Syntax

elementID.disabledImage

Possible Values

This attribute is a read/write String containing the image file name.

Remarks

The supported image formats are BMP, JPG, PNG, and GIF.

This image will be displayed whenever the disabled attribute of the control is set to true. If the disabled image is larger than the defined region, the disabled image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element

BUTTON.down

The down attribute specifies or retrieves a value indicating whether the BUTTON is in the up or down position.

Syntax

elementID.down

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2000-2003 Microsoft Corporation. All rights reserved.
true Indicates that the **BUTTON** is in the down position.
false Default. Indicates that the **BUTTON** is in the up position.

**Remarks**

In order for a **BUTTON** to remain in the down position, **sticky** must be set to true. By default, **sticky** is false, and any attempt to set **down** to true will be ignored.

If an invalid value is specified, the previous state is maintained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**
- **BUTTON.downImage**
- **BUTTON.downToolTip**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**BUTTON.downImage**

The **downImage** attribute specifies or retrieves the image representing the down state of the **BUTTON**.

**Syntax**

```
elementID.downImage
```

**Possible Values**

This attribute is a read/write **String** containing the image file name.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF.

The default image is the one specified in the **image** attribute, or its default.
If the down image is larger than the defined region in the ambient attribute, the down image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element
- BUTTON.down
- BUTTON.image

© 2000-2003 Microsoft Corporation. All rights reserved.

BUTTON.downToolTip

The downToolTip attribute specifies or retrieves the ToolTip text that appears when the mouse is over the BUTTON and the BUTTON is in the down state.

Syntax

elementID.downToolTip

Possible Values

This attribute is a read/write String with a default value of "" (empty string) and a maximum length of 1024 characters.

Remarks

When this attribute is set to "" (empty string), no ToolTip is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element
The `hoverDownImage` attribute specifies or retrieves the image displayed when the `BUTTON` is in the down state and the user hovers over it with the mouse pointer.

**Syntax**

```
elementType.hoverDownImage
```

**Possible Values**

This attribute is a read/write `String` containing the image file name.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF.

The default image is the one specified in the `downImage` attribute, or its default.

If the hover-down image is larger than the defined region in the ambient attribute, the hover-down image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `BUTTON Element`
- `BUTTON.downImage`
The **hoverImage** attribute specifies or retrieves the image displayed when the **BUTTON** is in the up state and the user hovers over it with the mouse pointer.

**Syntax**

```
elementID.hoverImage
```

**Possible Values**

This attribute is a read/write **String** containing the image file name.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF.

The default image is the one specified in the **image** attribute, or its default.

If the hover image is larger than the defined region in the ambient attribute, the hover image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**
- **BUTTON.image**
**BUTTON.image**

The **image** attribute specifies or retrieves the default image of the **BUTTON**.

**Syntax**

```
<elementID>.image
```

**Possible Values**

This attribute is a read/write **String** containing the image file name.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF (including animated GIFs).

If the **BUTTON** image is larger than the region defined by the **width** and **height** attributes, the image will be cropped.

If the image is not specified but the **height** and **width** are, then the image directly behind this control is displayed. This can facilitate drawing the image on the **VIEW** itself, reducing the number of separate image files needed.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**

---

**BUTTON.sticky**

The **sticky** attribute specifies or retrieves a value indicating whether the **BUTTON** is a toggle, that is, whether it is a two-state or single-state **BUTTON**.
Syntax

elementID.sticky

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>BUTTON is sticky.</td>
</tr>
<tr>
<td>false</td>
<td>Default. BUTTON is not sticky.</td>
</tr>
</tbody>
</table>

Remarks

If sticky is set to true, the BUTTON will change to the down state when clicked and will remain in that state until clicked again. When the BUTTON is in the down state, the down attribute will be true and the downImage will be displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element
- BUTTON.down
- BUTTON.downImage

© 2000-2003 Microsoft Corporation. All rights reserved.

BUTTON.tiled

The tiled attribute specifies or retrieves a value indicating whether the BUTTON image will be tiled.

Syntax

elementID.tiled

Possible Values
This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Image will be tiled.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Image will not be tiled.</td>
</tr>
</tbody>
</table>

Remarks

If the image is smaller than the actual region of the control, then tiling the image will repeat it until it fills the entire region. If an image is not specified or cannot be retrieved, tiled will be set to false.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element
- BUTTON.image

BUTTON.transparencyColor

The transparencyColor attribute specifies or retrieves the color that will be transparent in the BUTTON images.

Syntax

`elementID.transparencyColor`

Possible Values

This attribute is a read/write String with no default containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>The color of the pixel at location 0,0 in the image becomes the transparent color.</td>
</tr>
</tbody>
</table>
Remarks

A transparent color in an image allows whatever is behind the image to show through the transparent areas. The BUTTON will still receive clicks on the transparent region.

The transparent color can be any Internet Explorer color value. If the value of the transparencyColor attribute is set to "Auto", the color of the pixel at location 0,0 in the image is used.

If the color specified is not one of the valid IE colors, the previous value is maintained.

Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended when transparencyColor is used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element
- BUTTON.image
- Color Reference

BUTTON.upToolTip

The upToolTip attribute specifies or retrieves the ToolTip text that appears when the mouse is over the BUTTON and the BUTTON is in the up state.

Syntax

elementID.upToolTip

Possible Values
This attribute is a read/write **String** with a default value of "" (empty string) and a maximum length of 1024 characters.

**Remarks**

When this attribute is set to "" (empty string), no ToolTip is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**
- **BUTTON.down**

---

**CLOSEBUTTON**

This is a predefined **BUTTON** with the following default values.

```html
onclick="jscript: view.close();"
upToolTip="Close"
```

**Remarks**

This creates a **BUTTON** control that closes the Player. The ToolTips are localized. All properties of this **BUTTON** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player 7.0 or later.

**See Also**

- **BUTTON Element**
FFWDBUTTON

This is a predefined BUTTON with the following default values.

onclick="javascript:player.controls.fastForward()"
upToolTip="Fast Forward"
cursor="system"
enabled="wmEnabled:player.controls.fastForward"

Remarks

All properties of this BUTTON can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element

IMAGEBUTTON

This is a predefined BUTTON with the following default value.

cursor="Hand"

Remarks

All buttons can contain images. This element is useful for code clarification purposes to indicate a button that is used primarily as an image container.
All properties of this **BUTTON** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**MINIMIZEBUTTON**

This is a predefined **BUTTON** with the following default values.

```
onclick="jscript:view.minimize();"
upToolTip="Minimize"
```

**Remarks**

This will create a **BUTTON** control that will minimize the Player. The ToolTips are localized. All properties of this **BUTTON** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player 7.0 or later.

**See Also**

- **BUTTON Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.
MUTEBUTTON

This is a predefined BUTTON with the following default values.

```javascript
onclick="jscript:player.settings.mute=down;"
upToolTip="Mute"
downToolTip="Sound"
down="wmpprop:player.settings.mute"
sticky="true"
```

Remarks

This creates a BUTTON control that will mute and un-mute the audio. The ToolTips are localized. All properties of this BUTTON can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- **BUTTON Element**

NEXTBUTTON

This is a predefined BUTTON with the following default values.

```javascript
onclick="jscript:player.controls.next()"
upToolTip="Next"
cursor="system"
enabled="wmpproperties:player.controls.next"
```

Remarks
All properties of this **BUTTON** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**PAUSEBUTTON**

This is a predefined **BUTTON** with the following default values.

```html
onclick="jscript:player.controls.pause()"
upToolTip="Pause"
cursor="system"
enabled="wmpenabled:player.controls.pause"
```

**Remarks**

All properties of this **BUTTON** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTON Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.
PLAYBUTTON

This is a predefined BUTTON with the following default values.

onclick="javascript:player.controls.play()"
upToolTip="Play"
cursor="system"
enabled="wmpenabled:player.controls.play"

Remarks

All properties of this BUTTON can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element

PREVBUTTON

This is a predefined BUTTON with the following default values.

onclick="javascript:player.controls.previous()"
upToolTip="Previous"
cursor="system"
enabled="wmpenabled:player.controls.previous"

Remarks

All properties of this BUTTON can be overridden by explicitly specifying them.
Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTON Element**

REPEATBUTTON

This is a predefined **BUTTON** with the following default values.

```html
onclick="javascript:player.settings.setMode('loop',down);"
upToolTip="Turn Repeat On"
downToolTip="Turn Repeat Off"
down="wmpprop:player.settings.getMode('loop')"
sticky="true"
```

Remarks

This creates a **BUTTON** control that toggles Repeat on and off. The ToolTips are localized. All properties of this **BUTTON** can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- **BUTTON Element**
RETURNBUTTON

This is a predefined BUTTON with the following default values.

onclick="jscript:view.returnToMediaCenter();"
upToolTip="Return to Full Mode"

Remarks

This creates a BUTTON control that returns the Player to the media center. The ToolTips are localized. All properties of this BUTTON can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- BUTTON Element

REWBUTTON

This is a predefined BUTTON with the following default values.

onclick="jscript:player.controls.fastReverse()"
upToolTip="Fast Reverse"
cursor="system"
enabled="wmpenabled:player.controls.fastReverse"

Remarks

All properties of this BUTTON can be overridden by explicitly specifying them.
Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTON Element**

SHUFFLEBUTTON

This is a predefined BUTTON with the following default values.

```javascript
onclick="jscript:player.settings.setMode('shuffle',down);"
upToolTip="Turn Shuffle On"
downToolTip="Turn Shuffle Off"
down="wmpprop:player.settings.getMode('shuffle')"
sticky="true"
```

Remarks

This creates a **BUTTON** control that toggles Shuffle on and off. The ToolTips are localized. All properties of this **BUTTON** can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- **BUTTON Element**
STOPBUTTON

This is a predefined BUTTON with the following default values.

onclick="jscript:player.controls.stop()"
upToolTip="Stop"
cursor="system"
enabled="wmpenabled:player.controls.stop"

Remarks

All properties of this BUTTON can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTON Element

© 2000-2003 Microsoft Corporation. All rights reserved.

BUTTONELEMENT Element

The BUTTONELEMENT element defines a single button within a parent BUTTONGROUP element. It supports the following attributes. Predefined BUTTONELEMENT elements are also provided for convenience.

The BUTTONELEMENT element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td>Specifies or retrieves the value of the BUTTONELEMENT cursor that appears when the mouse is over the BUTTONELEMENT.</td>
</tr>
</tbody>
</table>
The `BUTTONELEMENT` element supports the following method.

**Method** | **Description**
---|---
`click` | Calls the `onclick` event handler defined for the `BUTTONELEMENT`.

The `BUTTONELEMENT` element can implement the ambient event handlers. For more information, see [Ambient Event Handlers](#). The `BUTTONELEMENT` element supports the following ambient attributes: `enabled` and `tabStop`.

Predefined effects are normal `EFFECTS` elements with various common attribute settings specified by default. The following predefined `BUTTONELEMENT` elements are available.

**Predefined BUTTONELEMENT** | **Description**
---|---
`FFWDELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls `player.controls.fastForward`.
`NEXTELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls `player.controls.next`.
`PAUSEELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls `player.controls.pause`.
`PLAYELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls `player.controls.play`.
`PREVELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls `player.controls.previous`.
`REWELEMENT` | A `BUTTONELEMENT` with a built in `onclick` event handler that calls
The `click` method calls the `onclick` event handler defined for the `BUTTONELEMENT`.

**Syntax**

`elementID.click()`

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

Use this method to provide an alternate means of running the code associated with a button.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `BUTTONELEMENT Element`
The `cursor` attribute specifies or retrieves the value of the `BUTTONELEMENT` cursor that appears when the mouse is over the `BUTTONELEMENT`.

**Syntax**

```plaintext
elementID.cursor
```

**Possible Values**

This attribute is a read/write `String`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Default. Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
</tbody>
</table>

**Remarks**

If an invalid value is specified, the previous value is maintained.

Cursor file name paths are ignored, so the cursor file must reside in the default directory.

**Requirements**
The `down` attribute specifies or retrieves a value indicating whether the button element is in the up or down position.

**Syntax**

```html
elementID.down
```

**Possible Values**

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Indicates the BUTTONELEMENT is in the down position.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Indicates the BUTTONELEMENT is in the up position.</td>
</tr>
</tbody>
</table>

**Remarks**

In order for a button element to remain in the down position, `sticky` must be set to true. By default, `sticky` is false, and any attempt to set `down` to true will be ignored.

If an invalid value is specified, the previous state is maintained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- BUTTONELEMENT Element

---

© 2000-2003 Microsoft Corporation. All rights reserved.
The `downToolTip` attribute specifies or retrieves the ToolTip text that appears when the mouse is over the `BUTTON ELEMENT` and the `BUTTON ELEMENT` is in the down state.

**Syntax**

```
elementID.downToolTip
```

**Possible Values**

This attribute is a read/write `String` with a default value of "" (empty string) with a maximum length of 1024 characters.

**Remarks**

When this attribute is set to "" (empty string), no ToolTip is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `BUTTON ELEMENT Element`
- `BUTTON ELEMENT.down`
**BUTTONELEMENT.index**

The `index` attribute retrieves the index of the BUTTONELEMENT within the BUTTONGROUP.

**Syntax**

```
elementID.index
```

**Possible Values**

This attribute is a read-only **Number (long)**.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- BUTTONELEMENT Element

---

**BUTTONELEMENT.mappingColor**

The `mappingColor` attribute specifies or retrieves the color key that identifies this BUTTONELEMENT in the BUTTONGROUP.

**Syntax**

```
elementID.mappingColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has no default value.

**Example**

The following sample is a complete skin definition file that illustrates how some of the BUTTONELEMENT
attributes are used. It can be found in the Samples directory that was installed with the SDK.

```xml
<THEME>
  <VIEW
    backgroundImage = "background.bmp"
    titleBar = "False"
  >
    <PLAYER
      URL = "http://proseware.com/mellow.wma"
    />
    <EFFECTS
      id = "myeffects"
      top = "25"
      left = "88"
      width = "180"
      height = "150"
    />
    <BUTTONGROUP
      mappingImage = "map.bmp"
      hoverImage = "hover.bmp"
    >
      <BUTTONELEMENT
        mappingColor = "#00FF00"
        upToolTip = "Next"
        onClick = "JScript:myeffects.next();"
      />
      <BUTTONELEMENT
        mappingColor = "#FF0000"
        upToolTip = "Previous"
        onClick = "JScript:myeffects.previous();"
      />
    </BUTTONGROUP>
  </VIEW>
</THEME>

Remarks

This attribute specifies the color of the region in the button group mappingImage that corresponds to this button element. All clicks on this region are handled by this button element.

If an invalid color is specified, the BUTTONELEMENT is not activated.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Color Reference
- BUTTONELEMENT Element
- BUTTONGROUP.mappingImage

© 2000-2003 Microsoft Corporation. All rights reserved.
The **sticky** attribute specifies or retrieves a value indicating whether the **BUTTONELEMENT** is a toggle, that is, whether it is a two-state or single-state button.

**Syntax**

```
(elementID).sticky
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td><strong>BUTTONELEMENT</strong> is sticky.</td>
</tr>
<tr>
<td>false</td>
<td>Default. <strong>BUTTONELEMENT</strong> is not sticky.</td>
</tr>
</tbody>
</table>

**Remarks**

If the **sticky** attribute is set to true, the button element will change to the down state when clicked and will remain in that state until clicked again. When the button element is in the down state, the **down** attribute will be true and the appropriate portion of the button group **downImage** will be displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTONELEMENT** Element
- **BUTTONELEMENT.down**
- **BUTTONGROUP.downImage**
BUTTONELEMENT.upToolTip

The `upToolTip` attribute specifies or retrieves the ToolTip text that appears when the mouse is over the BUTTONELEMENT and the BUTTONELEMENT is in the up state.

**Syntax**

```plaintext
elementID.upToolTip
```

**Possible Values**

This attribute is a read/write `String` with a default value of `""` (empty string) with a maximum length of 1024 characters.

**Example**

See the `mappingColor` attribute for a sample illustrating the use of this attribute.

**Remarks**

When this attribute is set to `""` (empty string), no ToolTip is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- BUTTONELEMENT Element

FFWDELEMENT

This is a predefined BUTTONELEMENT with the following default values.

```javascript
onclick="jscript:player.controls.fastForward()"
upToolTip="Fast Forward"
cursor="system"
enabled="wmpenabled:player.controls.fastForward"
```
Remarks

All properties of this **BUTTONELEMENT** can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTONELEMENT Element**

NEXTELEMENT

This is a predefined **BUTTONELEMENT** with the following default values.

```javascript
onclick="jscript:player.controls.next()"
upToolTip="Next"
cursor="system"
enabled="wmpenabled:player.controls.next"
```

Remarks

All properties of this **BUTTONELEMENT** can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTONELEMENT Element**
This is a predefined **BUTTONELEMENT** with the following default values.

```javascript
onclick="jscript:player.controls.pause()"
upToolTip="Pause"
cursor="system"
enabled="wmpenabled:player.controls.pause"
```

**Remarks**

All properties of this **BUTTONELEMENT** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTONELEMENT Element**

---

This is a predefined **BUTTONELEMENT** with the following default values.

```javascript
onclick="jscript:player.controls.play()"
upToolTip="Play"
cursor="system"
enabled="wmpenabled:player.controls.play"
```

**Remarks**

All properties of this **BUTTONELEMENT** can be overridden by explicitly specifying them.
Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTONELEMENT** Element

PREVELEMENT

This is a predefined **BUTTONELEMENT** with the following default values.

```javascript
onclick="jscript:player.controls.previous()"
upToolTip="Previous"
cursor="system"
enabled="wmpenabled:player.controls.previous"
```

Remarks

All properties of this **BUTTONELEMENT** can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTONELEMENT** Element
REWELEMENT

This is a predefined BUTTONELEMENT with the following default values.

onclick="javascript:player.controls.fastReverse()"
upToolTip="Fast Reverse"
cursor="system"
enabled="wmpenabled:player.controls.fastReverse"

Remarks

All properties of this BUTTONELEMENT can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.

See Also

• BUTTONELEMENT Element

STOPELEMENT

This is a predefined BUTTONELEMENT with the following default values.

onclick="javascript:player.controls.stop()"
upToolTip="Stop"
cursor="system"
enabled="wmpenabled:player.controls.stop"

Remarks

All properties of this BUTTONELEMENT can be overridden by explicitly specifying them.

Requirements

Windows Media Player version 7.0 or later.
The **BUTTONGROUP** element provides a way to group several buttons within a skin. These buttons can be specified by using **BUTTONELEMENT** elements as children of the **BUTTONGROUP** element.

The **BUTTONGROUP** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttonCount</td>
<td>Retrieves the number of buttons in the <strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>cursor</td>
<td>Specifies or retrieves the type of cursor that appears when the mouse is</td>
</tr>
<tr>
<td></td>
<td>over a button in the <strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>disabledImage</td>
<td>Specifies or retrieves the name of the image representing the disabled</td>
</tr>
<tr>
<td></td>
<td>state of the buttons in the <strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>downImage</td>
<td>Specifies or retrieves the name of the image representing the down state</td>
</tr>
<tr>
<td></td>
<td>of the <strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>hoverDownImage</td>
<td>Specifies or retrieves the name of the image representing the hover-down</td>
</tr>
<tr>
<td></td>
<td>state of a button in the <strong>BUTTONGROUP</strong>. The hover-down state occurs when</td>
</tr>
<tr>
<td></td>
<td>the button is in the down state and the user hovers over it with the mouse.</td>
</tr>
<tr>
<td>hoverImage</td>
<td>Specifies or retrieves the name of the image representing the hover state</td>
</tr>
<tr>
<td></td>
<td>of a button in the <strong>BUTTONGROUP</strong>. The hover state occurs when the button</td>
</tr>
<tr>
<td></td>
<td>is in the up state and the user hovers over it with the mouse.</td>
</tr>
<tr>
<td>hueShift</td>
<td>Specifies or retrieves the amount by which the hue of the <strong>BUTTONGROUP</strong></td>
</tr>
<tr>
<td></td>
<td>images is shifted.</td>
</tr>
<tr>
<td>image</td>
<td>Specifies or retrieves the name of the image representing the buttons of a</td>
</tr>
<tr>
<td></td>
<td><strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>mappingImage</td>
<td>Specifies or retrieves the name of the image representing the button map</td>
</tr>
<tr>
<td></td>
<td>of the <strong>BUTTONGROUP</strong>.</td>
</tr>
<tr>
<td>radio</td>
<td>Specifies or retrieves a value indicating whether the <strong>BUTTONGROUP</strong></td>
</tr>
</tbody>
</table>
The BUTTONGROUP element supports the following methods.

**Method** | **Description**
--- | ---
**click** | Calls the onclick event handler defined for the BUTTONELEMENT with the specified index.
**getButton** | Retrieves the BUTTONELEMENT with the specified index.

The BUTTONGROUP element supports the ambient attributes and can implement the ambient event handlers. For more information, see Ambient Attributes and Ambient Event Handlers.

See Also

- Skin Programming Reference

---

**BUTTONGROUP.buttonCount**

The **buttonCount** attribute retrieves the number of buttons in the BUTTONGROUP.

**Syntax**

elementID.buttonCount

**Possible Values**

This attribute is a read-only Number (long).
BUTTONGROUP.click

The click method calls the onclick event handler defined for the BUTTONELEMENT with the specified index.

Syntax

elementID.click(button)

Parameters

button

Number (long) containing the index of the BUTTONELEMENT with the onclick event handler to be called.

Return Values

This method does not return a value.

Remarks

Use this method to provide an alternate means of running the code associated with a button contained within the BUTTONGROUP.

Requirements

Windows Media Player 9 Series or later.

See Also

- BUTTONGROUP Element
The `cursor` attribute specifies or retrieves the type of cursor that appears when the mouse is over a button in the `BUTTONGROUP`.

**Syntax**

```plaintext
elementID.cursor
```

**Possible Values**

This attribute is a read/write `String` containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Default. Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
</tbody>
</table>

**Remarks**

The cursor specified applies to all buttons in the `BUTTONGROUP`. 
If you specify an invalid cursor value, it remains at the previously set value.

Cursor file name paths are ignored, so the cursor file must reside in the default directory.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTONGROUP Element

BUTTONGROUP.disabledImage

The disabledImage attribute specifies or retrieves the name of the image representing the disabled state of the buttons in the BUTTONGROUP.

Syntax

`elementID.disabledImage`

Possible Values

This attribute is a read/write String.

Remarks

The supported image formats are BMP, JPG, PNG, and GIF. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the hueShift and saturation attributes.

When the disabled attribute of a BUTTONELEMENT element is set to true, the corresponding region of the disabledImage for the BUTTONGROUP is displayed. If the disabled image is larger than the defined region, the disabled image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

Requirements

Windows Media Player version 7.0 or later.
See Also

- **BUTTONGROUP Element**
- **BUTTONGROUP.hueShift**
- **BUTTONGROUP.saturation**

**BUTTONGROUP.downImage**

The **downImage** attribute specifies or retrieves the name of the image representing the down state of the **BUTTONGROUP**.

**Syntax**

`elementID.downImage`

**Possible Values**

This attribute is a read/write **String**.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the **hueShift** and **saturation** attributes.

The default image is the one specified in the **image** attribute.

If the down image is larger than the defined region, the down image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTONGROUP Element**
- **BUTTONGROUP.hueShift**
- **BUTTONGROUP.image**
The `getButton` method retrieves the `BUTTONELEMENT` with the specified index.

**Syntax**

`elementID.getButton(button)`

**Parameters**

`button`

Number (`long`) containing the index of the `BUTTONELEMENT` to retrieve.

**Return Values**

This method returns an `Object` corresponding to a `BUTTONELEMENT` element.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `BUTTONGROUP Element`
- `BUTTONELEMENT.index`
BUTTONGROUP.hoverDownImage

The **hoverDownImage** attribute specifies or retrieves the name of the image representing the hover-down state of a button in the **BUTTONGROUP**. The hover-down state occurs when the button is in the down state and the user hovers over it with the mouse.

**Syntax**

```html
elementID.hoverDownImage
```

**Possible Values**

This attribute is a read/write **String**.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the **hueShift** and **saturation** attributes.

The default image is the one specified in the **downImage** attribute, or its default.

If the hover-down image is larger than the defined region, the hover-down image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **BUTTONGROUP Element**
- **BUTTONGROUP.downImage**
- **BUTTONGROUP.hueShift**
- **BUTTONGROUP.saturation**

© 2000-2003 Microsoft Corporation. All rights reserved.
The `hoverImage` attribute specifies or retrieves the name of the image representing the hover state of a button in the `BUTTONGROUP`. The hover state occurs when the button is in the up state and the user hovers over it with the mouse.

**Syntax**

```
elementID.hoverImage
```

**Possible Values**

This attribute is a read/write `String`.

**Remarks**

The supported image formats are BMP, JPG, PNG, and GIF. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the `hueShift` and `saturation` attributes.

The default image is the one specified in the `image` attribute, or its default.

If the hover image is larger than the defined region, the hover image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

**Example**

See the `BUTTONELEMENT.mappingColor` attribute for a sample illustrating the use of this attribute.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `BUTTONGROUP Element`
- `BUTTONGROUP.hueShift`
- `BUTTONGROUP.image`
- `BUTTONGROUP.saturation`

© 2000-2003 Microsoft Corporation. All rights reserved.
The `hueShift` attribute specifies or retrieves the amount by which the hue of the **BUTTONGROUP** images is shifted.

**Syntax**

\[elementID.hueShift\]

**Possible Values**

This attribute is a read/write *Number (float)* with a value ranging from 0.0 to 360.0 with a default value of 0.0.

**Remarks**

This attribute changes the hue value of the images specified by the `disabledImage`, `downImage`, `hoverDownImage`, `hoverImage`, and `image` attributes if they have been specified and they refer to 8-bit BMP images.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- **BUTTONGROUP Element**
- **BUTTONGROUP.disabledImage**
- **BUTTONGROUP.downImage**
- **BUTTONGROUP.hoverDownImage**
- **BUTTONGROUP.hoverImage**
- **BUTTONGROUP.image**
- **BUTTONGROUP.saturation**

---

The `image` attribute specifies or retrieves the name of the image representing the buttons of a **BUTTONGROUP**.

**Syntax**

\[elementID.image\]
Possible Values

This attribute is a read/write String.

Remarks

The supported image formats are BMP, JPG, PNG, and GIF. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the hueShift and saturation attributes.

If the image of the control is larger than the defined region, the image will be cropped.

If the image cannot be retrieved, a default image (the red-x image) is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTONGROUP Element
- BUTTONGROUP.hueShift
- BUTTONGROUP.saturation

© 2000-2003 Microsoft Corporation. All rights reserved.

BUTTONGROUP.mappingImage

The mappingImage attribute specifies or retrieves the name of the image representing the button map of the BUTTONGROUP.

Syntax

`elementID.mappingImage`

Possible Values

This attribute is a read/write String.

Remarks

This is a mandatory attribute that must be specified.
The mapping image should be the same dimensions as the main image. It is a map of the clickable areas of the main image. Construct the map by filling each clickable area with a different color.

When defining each BUTTONELEMENT, designate its color from the map using the mappingColor attribute. For example, if you have a stop-sign-shaped button graphic in the main image, you can create a map with the area of the sign colored red. The BUTTONELEMENT attribute must then be specified as red to make the stop-sign image clickable.

The supported image formats are BMP, JPG, PNG, and GIF (not including animated GIFs). Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended for mapping images.

Example

The following is an example of a mappingImage and the visible image it corresponds to. These images are part of the skin in the Tiny folder included with the SDK.

![Mapping Image Example](image)

See the BUTTONELEMENT.mappingColor attribute for a code sample illustrating the use of this attribute.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTONELEMENT.mappingColor
- BUTTONGROUP Element
- BUTTONGROUP.image
- BUTTONGROUP.transparencyColor

© 2000-2003 Microsoft Corporation. All rights reserved.

---

The radio attribute specifies or retrieves a value indicating whether the BUTTONGROUP is composed of radio buttons.
Syntax

```
$elementID$.radio
```

Possible Values

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The <strong>BUTTONGROUP</strong> is radio style.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The <strong>BUTTONGROUP</strong> is not radio style.</td>
</tr>
</tbody>
</table>

Remarks

If `radio` is set to true, all the **BUTTONELEMENT** elements in the **BUTTONGROUP** will be sticky, but only one button at a time will be in the down state.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **BUTTONELEMENT.sticky**
- **BUTTONGROUP Element**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**BUTTONGROUP.saturation**

The **saturation** attribute specifies or retrieves the saturation value of the **BUTTONGROUP** images.

Syntax

```
$elementID$.saturation
```

Possible Values

This attribute is a read/write **Number (float)** with a value ranging from 0.0 to 2.0 with a default value of 1.0.
Remarks

This attribute changes the saturation value of the images specified by the disabledImage, downImage, hoverDownImage, hoverImage, and image attributes if they have been specified and they refer to 8-bit BMP images.

Requirements

Windows Media Player 9 Series or later.

See Also

- BUTTONGROUP Element
- BUTTONGROUP.disabledImage
- BUTTONGROUP.downImage
- BUTTONGROUP.hoverDownImage
- BUTTONGROUP.hoverImage
- BUTTONGROUP.hueShift
- BUTTONGROUP.image

BUTTONGROUP.showBackground

The showBackground attribute specifies or retrieves a value indicating whether the BUTTONGROUP displays only the buttons, or displays the full bitmap specified in the image attribute.

Syntax

elementID.showBackground

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Buttons are displayed and the area not occupied by buttons is drawn from the Image bitmap.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Only the buttons are displayed.</td>
</tr>
</tbody>
</table>
Remarks

When `showBackground` is true, the entire main `image` will be visible.

When `showBackground` is false, only the areas corresponding to assigned `mappingImage` colors will be rendered. In other words, only `BUTTONELEMENT`s with their `mappingColor` assigned will be visible.

If an invalid value is specified, the previous state is maintained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `BUTTONGROUP Element`
- `BUTTONELEMENT.mappingColor`
- `BUTTONGROUP.image`
- `BUTTONGROUP.mappingImage`

© 2000-2003 Microsoft Corporation. All rights reserved.

**BUTTONGROUP.transparencyColor**

The `transparencyColor` attribute specifies or retrieves the transparent color of the `BUTTONGROUP` images.

**Syntax**

`elementID.transparencyColor`

**Possible Values**

This attribute is a read/write `String` with no default, containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>The pixel at location 0,0 in the image becomes the transparent color.</td>
</tr>
<tr>
<td><code>any Microsoft Internet Explorer color value</code></td>
<td>An Internet Explorer color value becomes the transparent color (for example, &quot;red&quot; or &quot;#FF0000&quot;).</td>
</tr>
<tr>
<td>None</td>
<td>Default. No transparency.</td>
</tr>
</tbody>
</table>
Remarks

A transparent color in an image will allow whatever is behind the image to show through the areas of transparency. The transparent region is clickable unless clipped by the clippingImage tag.

The color can be any Microsoft Internet Explorer color value. If the value is Auto, then the color of the pixel at location 0,0 in the image is used.

If the color specified is not one of the valid Internet Explorer colors, a warning is returned and the previous value is maintained.

Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended when transparencyColor is used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- BUTTONGROUP Element
- Color Reference

COLUMN Element

The COLUMN element represents a column within a playlist control. COLUMN elements are always children of PLAYLIST elements.

The COLUMN element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnName</td>
<td>Specifies or retrieves a column name in the PLAYLIST control.</td>
</tr>
<tr>
<td>columnID</td>
<td>Specifies or retrieves a column ID in the PLAYLIST control.</td>
</tr>
<tr>
<td>columnResizeMode</td>
<td>Specifies or retrieves the resize mode for this column.</td>
</tr>
<tr>
<td>columnWidth</td>
<td>Specifies or retrieves a column width in the PLAYLIST control.</td>
</tr>
</tbody>
</table>
COLUMN.columnName

The columnName attribute specifies or retrieves a column name in the PLAYLIST control.

Syntax

elementID.columnName

Possible Values

This attribute is a read/write String.

Remarks

The column name is the friendly name that appears in the header of the column.

Requirements

Windows Media Player 9 Series or later.

See Also

- COLUMN Element
The `columnID` attribute specifies or retrieves a column ID in the `PLAYLIST` control.

**Syntax**

```
$elemen$ .columnID
```

**Possible Values**

This attribute is a read/write `String`.

**Remarks**

The `columnID` values are the same values used with the `getItemInfo` method on a `Media` object. A list can be obtained by using the `Media.getAttributeCount` property to determine the number of attributes available for a given `Media` object. Index numbers can then be used with the `Media.getAttributeName` method to determine the names of the attributes, which can in turn be passed to `Media.getItemInfo`. The `columnID` property can only be set to these values, with the exception of some values that may not be returned by `Media.getAttributeName`. These values are listed below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Displays the name of the <code>Media</code> object.</td>
</tr>
<tr>
<td>duration</td>
<td>Displays the duration of the <code>Media</code> object.</td>
</tr>
<tr>
<td>sourceURL</td>
<td>Displays the URL of the <code>Media</code> object.</td>
</tr>
<tr>
<td>status</td>
<td>Displays the status of copying files.</td>
</tr>
<tr>
<td>size</td>
<td>Displays the size of the file that the <code>Media</code> object represents.</td>
</tr>
<tr>
<td>extension</td>
<td>Displays the file name extension of the file that the <code>Media</code> object represents.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `COLUMN Element`
- `Media Object`
- `Media.getAttributeCount`
- `Media.getAttributeName`
- `Media.getItemInfo`

© 2000-2003 Microsoft Corporation. All rights reserved.
The `columnResizeMode` attribute specifies or retrieves the resize mode for this column.

**Syntax**

```
elementID.columnResizeMode
```

**Possible Values**

This attribute is a read/write `String` containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutosizeHeader</td>
<td>Default. The column resizes to accommodate all data in both the column and the header.</td>
</tr>
<tr>
<td>AutosizeData</td>
<td>The column resizes to accommodate all data in the column only.</td>
</tr>
<tr>
<td>Fixed</td>
<td>The column is a fixed size.</td>
</tr>
<tr>
<td>Stretches</td>
<td>The column resizes to use the remaining space in the <code>PLAYLIST</code> control after all other columns are resized.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [COLUMN Element](#)
COLUMNS.columnWidth

The `columnWidth` attribute specifies or retrieves a column width in the PLAYLIST control.

Syntax

```plaintext
elementID.columnWidth
```

Possible Values

This attribute is a read/write `Number (long)` representing the width of the column in pixels.

Remarks

The `columnResizeMode` property must be set to "fixed" for this property to work.

Requirements

Windows Media Player 9 Series or later.

See Also

- COLUMN Element

CONTROLS Element

The CONTROLS element enables the following attributes of the Controls object to be specified within a skin.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentAudioLanguage</td>
<td>Specifies or retrieves the locale identifier (LCID) of the audio language for playback.</td>
</tr>
<tr>
<td>currentAudioLanguageIndex</td>
<td>Specifies or retrieves the index that corresponds to the audio language for playback.</td>
</tr>
<tr>
<td>currentItem</td>
<td>Specifies or retrieves the current media item.</td>
</tr>
<tr>
<td>currentMarker</td>
<td>Specifies or retrieves the current marker number.</td>
</tr>
</tbody>
</table>
CUSTOMSLIDER Element

The CUSTOMSLIDER element provides a way to create and manipulate a slider control of any shape, such as a circular knob. The following tables list the attributes and event handlers supported by this element.

The CUSTOMSLIDER element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursor</code></td>
<td>Specifies or retrieves the value of the slider control cursor that appears when the mouse is over the slider.</td>
</tr>
<tr>
<td><code>disabledImage</code></td>
<td>Specifies or retrieves the image of the slider used when the slider is disabled.</td>
</tr>
<tr>
<td><code>downImage</code></td>
<td>Specifies or retrieves the down state image of the custom slider.</td>
</tr>
<tr>
<td><code>hoverImage</code></td>
<td>Specifies or retrieves the image that displays when the mouse is over the custom slider.</td>
</tr>
<tr>
<td><code>image</code></td>
<td>Specifies or retrieves the name of the file containing the images corresponding to the various states of the custom slider.</td>
</tr>
<tr>
<td><code>max</code></td>
<td>Specifies or retrieves the maximum value of the range defined by the custom slider.</td>
</tr>
<tr>
<td><code>min</code></td>
<td>Specifies or retrieves the minimum value of the range defined by the custom slider.</td>
</tr>
<tr>
<td><code>positionImage</code></td>
<td>Specifies or retrieves the image map used to determine which position...</td>
</tr>
</tbody>
</table>
The **CUSTOMSLIDER** element can implement the following event handlers.

### Event handler | Description
--- | ---
**onDragBegin** | Handles an event that occurs when the user clicks and holds the left mouse button down and begins to drag the mouse.

**onDragEnd** | Handles an event that occurs when the left mouse button is released after a dragging operation.

**onPositionChange** | Handles an event that occurs when the position of the slider changes as a result of the user clicking or dragging.

The **CUSTOMSLIDER** element supports the ambient attributes and can implement the ambient event handlers. For more information, see [Ambient Attributes](#) and [Ambient Event Handlers](#).

**See Also**
- [Skin Programming Reference](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**CUSTOMSLIDER.cursor**

The **cursor** attribute specifies or retrieves the value of the slider control cursor that appears when the mouse is over the slider.

#### Syntax

```
(elementID).cursor
```

#### Possible Values

This attribute is a read/write **String**.
### CUSTOMSLIDER.disabledImage

The **disabledImage** attribute specifies or retrieves the image of the slider used when the slider is disabled.

#### Syntax

```plaintext
(elementID).disabledImage
```

#### Possible Values

This attribute is a read/write **String** containing the name of an image file.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Default. Cursor is a hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
</tbody>
</table>
Remarks

This attribute is optional. If it not specified, the file specified in the `image` attribute will be used.

The `disabledImage` represents the disabled state of the `CUSTOMSLIDER` control. It can be a single image or a series of images representing the various states of the slider. If multiple images are used, they are arranged in the same way as the `image` file.

The supported image file types are BMP, JPG, PNG, and GIF (not including animated GIFs).

Requirements

Windows Media Player version 7.0 or later.

See Also

- `CUSTOMSLIDER` Element
- `CUSTOMSLIDER.image`

CUSTOMSLIDER.downImage

The `downImage` attribute specifies or retrieves the down state image of the custom slider.

Syntax

```
elementID.downImage
```

Possible Values

This attribute is a read/write `String` containing the name of an image file.

Remarks

This attribute is optional. If it not specified, the file specified in the `image` attribute will be used.

The `downImage` represents the down state of the `CUSTOMSLIDER` control. It can be a single image or a series of images representing the various states of the slider. If multiple images are used, they are arranged in the same way as the `image` file.
The supported image file types are BMP, JPG, PNG, and GIF (not including animated GIFs).

Requirements

Windows Media Player version 7.0 or later.

See Also

- CUSTOMSLIDER Element
- CUSTOMSLIDER.image

CUSTOMSLIDER.hoverImage

The hoverImage attribute specifies or retrieves the image that appears when the mouse is over the custom slider.

Syntax

```html
elementID.hoverImage
```

Possible Values

This attribute is a read/write String containing the name of an image file.

Remarks

This attribute is optional. If it not specified, the file specified in the image attribute will be used.

The hoverImage represents the hover state of the CUSTOMSLIDER control; that is, the state that is shown when the mouse cursor hovers over it. It can be a single image or a series of images representing the various states of the slider. If multiple images are used, they are arranged in the same way as the image file.

The supported image file types are BMP, JPG, PNG, and GIF (not including animated GIFs).

Requirements

Windows Media Player version 7.0 or later.

See Also
The `image` attribute specifies or retrieves the name of the file containing the images corresponding to the various states of the custom slider.

### Syntax

```
elementID.image
```

### Possible Values

This attribute is a read/write `String` containing the name of an image file.

### Remarks

The `image` attribute is required. It specifies an image file that consists of one or more sub-images, arranged either horizontally or vertically, representing the various states of the custom slider. Each sub-image must have the same dimensions as the `positionImage` or the custom slider will not work correctly. The height or width of the overall image must therefore be an even multiple of the height or width of the `positionImage`.

The supported image file types are BMP, JPG, PNG, and GIF (not including animated GIFs).

### Example

The following is an example of a custom slider image. The corresponding `positionImage` is shown in the example section of the `positionImage` property.

The `positionImage` attribute also contains a code sample illustrating how the attributes of the `CUSTOMSLIDER` element are used.

### Requirements

Windows Media Player version 7.0 or later.
CUSTOMSLIDER.max

The max attribute specifies or retrieves the maximum value of the range defined by the custom slider.

Syntax

elementID.max

Possible Values

This attribute is a read/write Number (float) with a default value of 100.

Remarks

The value of max must be greater than that of min.

Example

See the positionImage attribute for a sample illustrating how the attributes of the CUSTOMSLIDER element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- CUSTOMSLIDER Element
- CUSTOMSLIDER.positionImage
The **min** attribute specifies or retrieves the minimum value of the range defined by the custom slider.

**Syntax**

```html
elementID.min
```

**Possible Values**

This attribute is a read/write **Number** (**float**) with a default value of zero.

**Remarks**

The value of **min** must be less than **max**.

**Example**

See the **positionImage** attribute for a sample illustrating how the attributes of the **CUSTOMSLIDER** element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **CUSTOMSLIDER Element**
- **CUSTOMSLIDER.max**
The **onDragBegin** event handler handles an event that occurs when the user clicks and holds the left mouse button down and begins to drag the mouse.

**Syntax**

```javascript
onDragBegin
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [CUSTOMSLIDER Element](#)

---

---

**CUSTOMSLIDER.onDragEnd**

The **onDragEnd** event handler handles an event that occurs when the left mouse button is released after a dragging operation.

**Syntax**

```javascript
onDragEnd
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [CUSTOMSLIDER Element](#)
CUSTOMSLIDER.onPositionChange

The **onPositionChange** event handler handles an event that occurs when the position of the slider changes as a result of the user clicking or dragging.

**Syntax**

```javascript
onPositionChange
```

**Remarks**

If the position of the custom slider changes as a result of the **value** attribute being modified in script, this event is not fired. To accommodate this possibility, implement the **value_onchange** event handler instead.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [CUSTOMSLIDER Element](#)

CUSTOMSLIDER.positionImage

The **positionImage** attribute specifies or retrieves the image map used to determine which position image from the **image** file to display.

**Syntax**

```javascript
elementID.positionImage
```

**Possible Values**
This attribute is a read/write String containing the name of an image file.

Remarks

This attribute is required and must be specified.

The positionImage is not displayed. Instead, it serves as a map defining the clickable regions of the displayed image. The displayed image is one of the sub-images of the image file and represents the actual state of the slider. The positionImage includes a number of gray scale regions equal to the number of these sub-images. The sub-images must have the same dimensions as the positionImage or the custom slider will not work correctly.

Any region that is not in gray scale will not be clickable. The clickable regions should be set to color values that range evenly across the gray scale spectrum from black to white, with the first region being pure black and the last region being pure white. The color values of each successive region should be incremented by a value equal to 255 divided by the total number of regions minus one, rounding to the nearest whole number.

For example, if there are six regions, the increment would be 51 (255 divided by 5), and the six gray scale values would be 0, 51, 102, 153, 204, and 255. The hexadecimal color values for the six regions would then be #000000, #333333, #666666, #999999, #CCCCCC, and #FFFFFF.

In this way, the regions will have a sequence dictated by their gray scale color values, and this sequence will correspond to the sequence of sub-images in the image file. When one of the regions is clicked, the corresponding sub-image is shown and the value of the custom slider is updated accordingly.

The supported image file types are BMP, JPG, PNG, and GIF (not including animated GIFs).

Example

The following is an example of a custom slider positionImage. The corresponding image is shown in the example section of the image property.

![Example Image]

The following code illustrates the use of CUSTOMSLIDER attributes.

```xml
<THEME>
  <VIEW
    backgroundImage = "background.bmp"
    titleBar = "False"
  >

  <PLAYER
    URL = "http://proseware.com/mellow.wma"
  >
  <CONTROLS
    currentPosition_onchange = "myslider.value = player.controls.currentPosition;"
  />
</PLAYER>

<SLIDER
  id = "myslider"
  min = "0"
  max = "wmpprop:player.currentMedia.duration"
```
Requirements

Windows Media Player version 7.0 or later.

See Also
CUSTOMSLIDER.toolTip

The **toolTip** attribute specifies or retrieves the ToolTip text for the slider.

**Syntax**

```
elementID.toolTip
```

**Possible Values**

This attribute is a read/write **String** with a maximum length of 1024 characters. It has no default value.

**Remarks**

When this attribute is set to "" (empty string), no ToolTip is displayed.

**Example**

See the **positionImage** attribute for a sample illustrating how the attributes of the **CUSTOMSLIDER** element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **CUSTOMSLIDER Element**
The **transparencyColor** attribute specifies or retrieves the transparency color of the custom slider images.

### Syntax

```
elementID.transparencyColor
```

### Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value. It has no default value.

### Remarks

Any part of the custom slider images containing the **transparencyColor** will allow the background to show through.

Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended when **transparencyColor** is used.

### Example

See the **positionImage** attribute for a sample illustrating how the attributes of the **CUSTOMSLIDER** element are used.

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- Color Reference
- **CUSTOMSLIDER** Element
CUSTOMSLIDER.value

The value attribute specifies or retrieves the current position of the slider.

Syntax

elementID.value

Possible Values

This attribute is a read/write Number (float) with a default value equal to the min attribute.

Remarks

The value must be greater than or equal to min and less than or equal to max. If the value falls outside the range, a warning is issued, and the value does not change.

Example

See the positionImage attribute for a sample illustrating how the attributes of the CUSTOMSLIDER element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- CUSTOMSLIDER Element

EDITBOX Element

The EDITBOX element provides a way for users to enter text within a skin.

The EDITBOX element supports the following attributes.
The **EDITBOX** element supports the following methods.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Specifies or retrieves the background color for the edit box control.</td>
</tr>
<tr>
<td>border</td>
<td>Specifies or retrieves a value indicating whether the edit box control has a border.</td>
</tr>
<tr>
<td>editStyle</td>
<td>Specifies or retrieves the style of the edit box control.</td>
</tr>
<tr>
<td>fontFace</td>
<td>Specifies or retrieves the font used for text in the edit box control.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Specifies or retrieves the font size for the edit box control.</td>
</tr>
<tr>
<td>fontStyle</td>
<td>Specifies or retrieves the font style for the edit box control.</td>
</tr>
<tr>
<td>foregroundColor</td>
<td>Specifies or retrieves the text color in the edit box control.</td>
</tr>
<tr>
<td>justification</td>
<td>Specifies or retrieves the alignment of the text within the edit box control.</td>
</tr>
<tr>
<td>lineCount</td>
<td>Retrieves the number of lines in the edit box control.</td>
</tr>
<tr>
<td>readOnly</td>
<td>Specifies or retrieves a value indicating whether text in the edit box control is read-only or can be edited.</td>
</tr>
<tr>
<td>textLimit</td>
<td>Specifies or retrieves the maximum number of characters that the user can type in the edit box control.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies or retrieves the text that is displayed in the edit box control.</td>
</tr>
<tr>
<td>wordWrap</td>
<td>Specifies or retrieves a value indicating whether word wrap is enabled in the edit box control.</td>
</tr>
</tbody>
</table>

The **EDITBOX** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLine</td>
<td>Retrieves the text for the line with the specified index.</td>
</tr>
<tr>
<td>getLineFromChar</td>
<td>Retrieves the line index for the specified character index.</td>
</tr>
<tr>
<td>getLineIndex</td>
<td>Retrieves the character index for the specified line index.</td>
</tr>
<tr>
<td>getSelectionEnd</td>
<td>Retrieves the ending position of the selected text in the edit box control.</td>
</tr>
<tr>
<td>getSelectionStart</td>
<td>Retrieves the starting position of the selected text in the edit box control.</td>
</tr>
</tbody>
</table>
replaceSelection

Replaces the current selection with the specified text.

setSelection

Selects the text in the edit box control from the specified start index to the specified end index.

The EDITBOX element supports the ambient attributes and can implement all ambient event handlers with the exception of onclick. For more information, see Ambient Attributes and Ambient Event Handlers.

Note This element requires Windows Media Player for Windows XP or later.

See Also

- Skin Programming Reference

EDITBOX.backgroundColor

The backgroundColor attribute specifies or retrieves the background color for the edit box control.

Syntax

```
elementID.backgroundColor
```

Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value or the value "none". It has a default value equal to the background color in Windows.

Requirements

Windows Media Player for Windows XP or later.

See Also

- Color Reference
- EDITBOX Element
- EDITBOX.foregroundColor
EDITBOX.border

The border attribute specifies or retrieves a value indicating whether the edit box control has a border. Can only be set at design time.

Syntax

elementID.border

Possible Values

This attribute is a Boolean specified at design time and read-only thereafter. It has a default value of true.

Remarks

The border appears as a two-pixel-wide sunken border around the edit box control with system colors.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX Element

EDITBOX.editStyle

The editStyle attribute specifies or retrieves the style of the edit box control.
Syntax

elementID.editStyle

Possible Values

This attribute is a read/write String containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Default. Displays normal text on a single line.</td>
</tr>
<tr>
<td>password</td>
<td>Displays asterisks (*) in place of text. Can only be specified at design time.</td>
</tr>
<tr>
<td>uppercase</td>
<td>Displays text as all uppercase.</td>
</tr>
<tr>
<td>lowercase</td>
<td>Displays text as all lowercase.</td>
</tr>
<tr>
<td>number</td>
<td>Only displays numbers.</td>
</tr>
<tr>
<td>multiline</td>
<td>Displays multiple lines of text. Can only be specified at design time.</td>
</tr>
</tbody>
</table>

Remarks

This attribute can only be set to "password" or "multiline" at design time. If it is set to "multiline", the value cannot be changed at run time.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX Element

© 2000-2003 Microsoft Corporation. All rights reserved.

---

EDITBOX.fontFace

The fontFamily attribute specifies or retrieves the font for text in the edit box control.
Syntax

elementID.fontFace

Possible Values

This attribute is a read/write String.

Remarks

This attribute can be the name of any valid font available in Windows. Windows Media Player will not support installing fonts, so choose a font that you know will be on the intended system.

If the fontFace specified is not available on the user's system, the edit box control defaults to the Windows system font. The default value for English-language systems is "Tahoma". For international systems, the default is loaded from a resource file.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX.Element
- EDITBOX.fontSize
- EDITBOX.fontStyle

© 2000-2003 Microsoft Corporation. All rights reserved.

EDITBOX.fontSize

The fontSize attribute specifies or retrieves the font size for the edit box control.

Syntax

elementID.fontSize

Possible Values

This attribute is a read/write Number (long) specifying the font size in points. The default value is 10.

Requirements
The `fontStyle` attribute specifies or retrieves the font style for the edit box control.

### Syntax

```
elementID.fontStyle
```

### Possible Values

This attribute is a read/write `String` containing one or more of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold font style.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic font style.</td>
</tr>
<tr>
<td>Underline</td>
<td>Underline font style.</td>
</tr>
<tr>
<td>Strikeout</td>
<td>Strikeout font style.</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal font style.</td>
</tr>
</tbody>
</table>

### Remarks

Any combination of the values can be used, separated with spaces. The Normal style has priority over all other values, and any others specified along with Normal will be ignored.

For rendering purposes, Normal is the default font style. The default value retrieved from this attribute is "" (empty string).

### Requirements
EDITBOX.foregroundColor

The **foregroundColor** attribute specifies or retrieves the text color in the edit box control.

**Syntax**

```javascript
elementID.foregroundColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value equal to the Windows text color.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- [Color Reference](#)
- [EDITBOX Element](#)
- [EDITBOX.backgroundColor](#)
EDITBOX.getLine

The **getLine** method retrieves the text for the line with the specified index.

**Syntax**

```
elementID.getLine(index)
```

**Parameters**

*index*

**Number (long)** containing the index of the line to retrieve.

**Return Values**

This method returns a **String**.

**Remarks**

If the index is not valid, an empty string is returned. This method can only be called after the control becomes visible.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- EDITBOX Element
- EDITBOX.getLineFromChar
- EDITBOX.getLineIndex
EDITBOX.getLineFromChar

The `EDITBOX.getLineFromChar` method retrieves the line index for the specified character index.

Syntax

```java
elementID.getLineFromChar(index)
```

Parameters

`index`

Number (long) containing the index of the character whose line index is to be retrieved.

Return Values

This method returns a Number (long).

Remarks

If the specified character position is –1, this method retrieves the line index of the current line.

This method can only be called after the control becomes visible.

Requirements

Windows Media Player for Windows XP or later.

See Also

- `EDITBOX.Element`
- `EDITBOX.getLine`
- `EDITBOX.getLineIndex`

© 2000-2003 Microsoft Corporation. All rights reserved.

EDITBOX.getLineIndex

The `EDITBOX.getLineIndex` method retrieves the character index of the first character on the line with the specified line index.
Syntax

elementID.getLineIndex(index)

Parameters

index

Number (long) containing the index of the line whose character index is to be retrieved.

Return Values

This method returns a Number (long).

Remarks

If the specified line index is –1, the line containing the insertion point is used.

This method can only be called after the control becomes visible.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX Element
- EDITBOX.getLine
- EDITBOX.getLineFromChar

EDITBOX.getSelectionEnd

The getSelectionEnd method retrieves the ending position of the selected text in the editbox control.

Syntax

elementID.getSelectionEnd()
This method takes no parameters.

**Return Values**

This method returns a **Number (long)**.

**Remarks**

If no text is selected, this method returns the position of the insertion point.

If the control is multiline, this method returns the character index in the control, not the line index.

This method can only be called after the control becomes visible.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- **EDITBOX Element**
- **EDITBOX.getSelectionStart**
- **EDITBOX.replaceSelection**
- **EDITBOX.setSelection**

© 2000-2003 Microsoft Corporation. All rights reserved.

**EDITBOX.getSelectionStart**

The `getSelectionStart` method retrieves the starting position of the selected text in the editbox control.

**Syntax**

```javascript
elementID.getSelectionStart()
```

**Parameters**

This method takes no parameters.

**Return Values**
This method returns a **Number (long)**.

**Remarks**

If no text is selected, this method returns the position of the insertion point.

If the control is multiligne, this method returns the character index in the control, not the line index.

This method can only be called after the control becomes visible.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- EDITBOX Element
- EDITBOX.getSelectionEnd
- EDITBOX.replaceSelection
- EDITBOX.setSelection

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**EDITBOX.justification**

The **justification** attribute specifies or retrieves the alignment of the text within the editbox control.

**Syntax**

`elementID.justification`

**Possible Values**

This attribute is a read/write **String** containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Default. Aligns the text to the left edge of the edit box control.</td>
</tr>
<tr>
<td>Right</td>
<td>Aligns the text to the right edge of the edit box</td>
</tr>
</tbody>
</table>
Remarks

There is a margin of two pixels.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX Element

EDITBOX.lineCount

The lineCount attribute retrieves the number of lines in the edit box control.

Syntax

`elementID.lineCount`

Possible Values

This attribute is a read-only Number (long).

Remarks

This attribute is useful only when editStyle is set to "multiline".

Requirements

Windows Media Player for Windows XP or later.

See Also
EDITBOX.readOnly

The `readOnly` attribute specifies or retrieves a value indicating whether text in the edit box control is read-only or can be edited.

**Syntax**

`elementID.readOnly`

**Possible Values**

This attribute is a read/write `Boolean` with a default value of `false`.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- `EDITBOX Element`

EDITBOX.replaceSelection

The `replaceSelection` method replaces the current selection with the specified text.
Syntax

`elementID.replaceSelection(newValue)`

Parameters

`newValue`

String containing the text to replace the selected text.

Return Values

This method does not return a value.

Remarks

If there is no text selected, the replacement text is inserted at the current location of the insertion point.

This method can only be called after the control becomes visible.

Requirements

Windows Media Player for Windows XP or later.

See Also

- `EDITBOX.Element`
- `EDITBOX.getSelectionEnd`
- `EDITBOX.getSelectionStart`
- `EDITBOX.setSelection`

© 2000-2003 Microsoft Corporation. All rights reserved.
Parameters

start

Number (long) containing the character index of the starting position of the selected text.

dead

Number (long) containing the character index of the ending position of the selected text.

Return Values

This method does not return a value.

Remarks

If the start is 0 and the end is –1, all of the text in the edit box control is selected. If the start is –1, any current selection is deselected.

This method can only be called after the control becomes visible.

Requirements

Windows Media Player for Windows XP or later.

See Also

- EDITBOX Element
- EDITBOX.getSelectionEnd
- EDITBOX.getSelectionStart
- EDITBOX.replaceSelection

EDITBOX.textLimit

The textLimit attribute specifies or retrieves the maximum number of characters that the user can type in the edit box control.

Syntax
Possible Values

This attribute is a read/write **Number (long)** with a default value of 0, which indicates no limit.

Requirements

Windows Media Player for Windows XP or later.

See Also

- **EDITBOX Element**

EDITBOX.value

The **value** attribute specifies or retrieves the text that is displayed in the edit box control.

Syntax

`elementID.value`

Possible Values

This attribute is a read/write **String** with a default value of "" (empty string).

Requirements

Windows Media Player for Windows XP or later.

See Also

- **EDITBOX Element**
EDITBOX.wordWrap

The **wordWrap** attribute specifies or retrieves a value indicating whether word wrap is enabled.

**Syntax**

```
elementID.wordWrap
```

**Possible Values**

This attribute is a read/write **Boolean** with a default value of true.

**Remarks**

This attribute is useful only when **editStyle** is set to "multiline".

If word wrap is disabled, and the text does not fit in the control, the text is truncated.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- **EDITBOX Element**
- **EDITBOX.editStyle**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

EFFECTS Element

The **EFFECTS** element provides a way to organize and manipulate visualizations by using the following attributes and methods. A predefined **EFFECTS** element is also provided for convenience.
The **EFFECTS** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowAll</td>
<td>Specifies or retrieves a value indicating whether to include all the visualizations in the registry.</td>
</tr>
<tr>
<td>currentEffect</td>
<td>Specifies or retrieves the current visualization.</td>
</tr>
<tr>
<td>currentEffectPresetCount</td>
<td>Retrieves number of available presets for the current visualization.</td>
</tr>
<tr>
<td>currentEffectTitle</td>
<td>Retrieves the display title of the current visualization.</td>
</tr>
<tr>
<td>currentEffectType</td>
<td>Retrieves the registry name of the current visualization.</td>
</tr>
<tr>
<td>currentPreset</td>
<td>Specifies or retrieves the current preset of the current visualization.</td>
</tr>
<tr>
<td>currentPresetTitle</td>
<td>Retrieves the title of the current preset of the current visualization.</td>
</tr>
<tr>
<td>effectCanGoFullScreen</td>
<td>Retrieves a value indicating whether the current visualization can be displayed full-screen.</td>
</tr>
<tr>
<td>effectCount</td>
<td>Retrieves the number of visualizations available.</td>
</tr>
<tr>
<td>effectHasPropertyPage</td>
<td>Retrieves a value indicating whether the current visualization has a property page.</td>
</tr>
<tr>
<td>fullScreen</td>
<td>Specifies or retrieves a value indicating whether the current visualization is displayed full-screen. Can only be set at run time.</td>
</tr>
<tr>
<td>windowed</td>
<td>Specifies or retrieves a value indicating whether the visualization will be windowed or windowless, that is, whether the entire rectangle of the control will be visible at all times, or whether it can be clipped. Can only be set at design time.</td>
</tr>
</tbody>
</table>

The **EFFECTS** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>effectTitle</td>
<td>Retrieves the friendly name of the visualization with the specified registry index.</td>
</tr>
<tr>
<td>effectType</td>
<td>Retrieves the registry name of the visualization with the specified registry index.</td>
</tr>
<tr>
<td>next</td>
<td>Displays the next visualization preset, moving to the next visualization if necessary.</td>
</tr>
<tr>
<td>nextEffect</td>
<td>Displays the first preset of the next visualization, skipping intervening presets.</td>
</tr>
<tr>
<td>nextPreset</td>
<td>Displays the next preset of the current visualization.</td>
</tr>
<tr>
<td>previous</td>
<td>Displays the previous visualization preset, moving to the last preset of the previous visualization if necessary.</td>
</tr>
<tr>
<td>previousEffect</td>
<td>Displays the previous visualization, skipping presets.</td>
</tr>
</tbody>
</table>
The **EFFECTS** element supports the ambient attributes and can implement the ambient event handlers. For more information, see [Ambient Attributes](#) and [Ambient Event Handlers](#).

Predefined effects are normal **EFFECTS** elements with various common attribute settings specified by default. The following predefined effects are available.

<table>
<thead>
<tr>
<th>Predefined EFFECTS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMPEFFECTS</td>
<td>An <strong>EFFECTS</strong> element that iterates through the available effects.</td>
</tr>
</tbody>
</table>

See Also

- [Skin Programming Reference](#)

---

**EFFECTS.allowAll**

The **allowAll** attribute specifies or retrieves a value indicating whether to include all the visualizations that are in the registry.

**Syntax**

```
elementID.allowAll
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Allows cycling of all visualizations on the user's system.</td>
</tr>
<tr>
<td>false</td>
<td>Limits cycling to visualizations appearing within <strong>EFFECTS</strong> tags.</td>
</tr>
</tbody>
</table>

**Remarks**
If this attribute is set to false, only the visualizations appearing within `EFFECTS` tags can be cycled through using previous/next. If it is set to true, then all visualizations that are registered on the user's system can be cycled through. If it is set to true and you specify any visualizations within `EFFECTS` tags, then the attributes specified in these tags are applied to all visualizations on the user's system.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `EFFECTS Element`

---

© 2000-2003 Microsoft Corporation. All rights reserved.

**EFFECTS.currentEffect**

The `currentEffect` attribute specifies or retrieves the current visualization.

**Syntax**

```
elementID.currentEffect
```

**Possible Values**

This attribute is a read/write `object`. The default value is the first visualization in the authoring order. If there are no visualizations authored in the skin, the default is the first visualization in the registry.

**Remarks**

You can use this object to access custom visualizations you have created. For example, you could expose a property through the `IDispatch` interface in your visualization. You can then change the property value from your skin by making your visualization the current effect, and then using the `currentEffect` object to set a new value for the property. For example, if your visualization exposes a property named `backgroundColor`, the following JScript code specifies a new value:

```
// The EFFECTS element ID is MyEffects.
MyEffects.currentEffect.backgroundColor = "blue";
```

**Requirements**

Windows Media Player version 7.0 or later.
The `currentEffectPresetCount` attribute retrieves number of available presets for the current visualization.

### Syntax

```
elementID.currentEffectPresetCount
```

### Possible Values

This attribute is a read-only `Number (long)`.

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- `EFFECTS Element`
- `EFFECTS.currentEffectTitle`
- `EFFECTS.currentEffectType`
The `currentEffectTitle` attribute retrieves the display title of the current visualization.

Syntax

```plaintext
elementID.currentEffectTitle
```

Possible Values

This attribute is a read-only `String`.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `EFFECTS.Element`
- `EFFECTS.currentEffect`
- `EFFECTS.effectTitle`

---

**EFFECTS.currentEffectType**

The `currentEffectType` attribute specifies or retrieves the registry name of the current visualization. This name is a unique ID defined by the visualization author.

Syntax

```plaintext
elementID.currentEffectType
```

Possible Values

This attribute is a read/write `String`.

Remarks

You can use this attribute at run time to change the currently displayed effect. To do this, follow these steps:

1. Use `effectCount` to retrieve the count of registered effects.
2. In a loop, retrieve the name of each registered effect by using `effectType`.
3. Specify one of the names you retrieved for `currentEffectType` to set the current effect.
Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element
- EFFECTS.currentEffect
- EFFECTS.effectCount
- EFFECTS.effectType

© 2000-2003 Microsoft Corporation. All rights reserved.

EFFECTS.currentPreset

The currentPreset attribute specifies or retrieves the current preset of the current visualization.

Syntax

elementID.currentPreset

Possible Values

This attribute is a read/write Number (long) indicating the index of the preset. The indexes begin with zero, which is also the default value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element

© 2000-2003 Microsoft Corporation. All rights reserved.
EFFECTS.currentPresetTitle

The `currentPresetTitle` attribute retrieves the title of the current preset of the current visualization.

**Syntax**

```
elementID.currentPresetTitle
```

**Possible Values**

This attribute is a read-only `String`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [EFFECTS Element](#)

---

EFFECTS.effectCanGoFullScreen

The `effectCanGoFullScreen` attribute retrieves a value indicating whether the current visualization can be displayed full-screen.

**Syntax**

```
elementID.effectCanGoFullScreen
```

**Possible Values**

This attribute is a read-only `Boolean`.

---

© 2000-2003 Microsoft Corporation. All rights reserved.
### EFFECTS.effectCount

The `effectCount` attribute retrieves the number of visualizations available.

#### Syntax

```
elementID.effectCount
```

#### Possible Values

This attribute is a read-only `Number (long)`.

#### Requirements

Windows Media Player for Windows XP or later.

#### See Also

- **EFFECTS Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

© 2000-2003 Microsoft Corporation. All rights reserved.
EFFECTS.effectHasPropertyPage

The `effectHasPropertyPage` attribute retrieves a value indicating whether the current visualization has a property page.

**Syntax**

```
elementID.effectHasPropertyPage
```

**Possible Values**

This attribute is a read-only `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Visualization has a property page.</td>
</tr>
<tr>
<td>false</td>
<td>Visualization does not have a property page.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [EFFECTS Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

EFFECTS.effectTitle

The `effectTitle` method retrieves the display title of the visualization with the specified registry index.
Syntax

```
elementID.effectTitle(index)
```

Parameters

`index`

**Number (long)** containing the registry index of a visualization.

Return Values

This method returns a **String**.

Remarks

This method is used for displaying visualization titles in a user interface.

Requirements

Windows Media Player 9 Series or later.

See Also

- **EFFECTS Element**
- **EFFECTS.currentEffectTitle**

EFFECTS.effectType

The `effectType` method retrieves the registry name of the visualization with the specified registry index. This name is a unique ID defined by the visualization author.

Syntax

```
elementID.effectType(index)
```

Parameters

`index`

© 2000-2003 Microsoft Corporation. All rights reserved.

© 2000-2003 Microsoft Corporation. All rights reserved.
Number (long) containing the registry index of a visualization.

Return Values

This method returns a String.

Remarks

This method is useful for switching between visualizations in script. A user interface could display the set of titles, but when the user selects one, the script must use currentEffectType to switch visualizations.

Requirements

Windows Media Player 9 Series or later.

See Also

- EFFECTS Element
- EFFECTS.currentEffectType

© 2000-2003 Microsoft Corporation. All rights reserved.

EFFECTS.fullScreen

The fullScreen attribute specifies or retrieves a value indicating whether the current visualization is displayed full-screen. This attribute can only be set at run time.

Syntax

`elementID.fullScreen`

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Visualization is displayed full-screen.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Visualization is not displayed full-screen.</td>
</tr>
</tbody>
</table>
Remarks

A visualization can be put into full-screen mode only while media is playing or paused. If Player.currentMedia is video, a video plug-in must be present. If it is audio, a visualization plug-in that supports full-screen display must be present.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element

EFFECTS.next

The next method displays the next visualization preset, moving to the next visualization if necessary.

Syntax

```
elementID.next()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

If the current preset is the last one in the series of all available visualizations, the first preset of the first visualization is made current.

Requirements

Windows Media Player version 7.0 or later.
EFFECTS.nextEffect

The **nextEffect** method displays the first preset of the next visualization, skipping intervening presets.

**Syntax**

```
elementType.nextEffect()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

This method displays the first preset of the next visualization in the authoring order. If the current visualization is the last in the authoring order, and if `allowAll` is false, the first visualization is made current.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- EFFECTS Element
- EFFECTS.allowAll
- EFFECTS.previousEffect
**EFFECTS.nextPreset**

The `nextPreset` method displays the next preset of the current visualization.

**Syntax**

```javascript
elementID.nextPreset()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

If the current preset is the last one in the list for the current visualization, the first preset is made current.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- EFFECTS Element
- EFFECTS.allowAll
- EFFECTS.previous
EFFECTS.previous

The `previous` method displays the previous visualization preset, moving to the last preset of the previous visualization if necessary.

Syntax

```javascript
elementID.previous()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

If the current preset is the first one in the series of all available visualizations, the last preset of the last visualization is made current.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS.Element
- EFFECTS.next
- EFFECTS.allowAll

EFFECTS.previousEffect

The `previousEffect` method displays the previous visualization, skipping presets.

Syntax

```javascript
elementID.previousEffect()
```

© 2000-2003 Microsoft Corporation. All rights reserved.
Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Remarks

This method displays the previous visualization in the authoring order. If the current visualization is the first in the authoring order, and if allowAll is false, the last visualization is made current.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element
- EFFECTS.nextEffect
- EFFECTS.allowAll

EFFECTS.previousPreset

The previousPreset method displays the previous preset of the current visualization.

Syntax

```
elementID.previousPreset()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.
Remarks

If the current preset is the first one in the list for the current visualization, the last preset is made current.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element
- EFFECTS.allowAll
- EFFECTS.previous

EFFECTS.settings

The settings method displays the attribute page for the current visualization, if present.

Syntax

```
elementID.settings()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EFFECTS Element
The `windowed` attribute specifies or retrieves a value indicating whether the visualization will be windowed or windowless, that is, whether the entire rectangle of the control will be visible at all times, or whether it can be clipped. This attribute can only be set at design time.

**Syntax**

```xml
elementID.windowed
```

**Possible Values**

This attribute is a **Boolean** specified at design time and read-only thereafter.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The control will be windowed.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The control will be windowless.</td>
</tr>
</tbody>
</table>

**Remarks**

If a non-rectangular visualization window is desired, or if any part of the window is covered by an image, this attribute must be set to `false`. This sacrifices some performance to do the necessary clipping.

If `windowed` is set to `true`, any image covering the visualization window is ignored, and the visualization window has the highest-level z-order.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **EFFECTS Element**
WMPEFFECTS

This is a predefined EFFECTS element with the following default values.

```xml
horizontalAlignment="stretch"
verticalAlignment="stretch"
height="200"
width="250"
tabStop="false"
onclick="next();"
```

**Remarks**

This will create an EFFECTS element that will step through the visualization presets when the user clicks the control. It will also stretch the visualizations when the player is resized.

The initial visualization preset shown is the one selected on the View menu under Visualizations. Changing the selection on this menu will automatically change the preset displayed by this element when the Player is in skin mode. The View menu is displayed in the full mode of the Player or when the VIEW.titleBar attribute is set to true in a skin.

All properties of this EFFECTS element can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player 7.0 or later.

**See Also**

- EFFECTS Element

© 2000-2003 Microsoft Corporation. All rights reserved.
The **EQUALIZERSETTINGS** element provides a way to manipulate the graphic equalizer and other audio settings of Windows Media Player using the attributes and method listed here.

The **EQUALIZERSETTINGS** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bands</strong></td>
<td>Retrieves the number of frequency bands supported.</td>
</tr>
<tr>
<td><strong>bypass</strong></td>
<td>Specifies or retrieves a value indicating whether the equalizer filter is bypassed in the filter graph.</td>
</tr>
<tr>
<td><strong>crossFade</strong></td>
<td>Specifies or retrieves a value indicating whether cross-fade is enabled.</td>
</tr>
<tr>
<td><strong>crossFadeWindow</strong></td>
<td>Specifies or retrieves the amount of cross-fade overlap in milliseconds.</td>
</tr>
<tr>
<td><strong>currentPreset</strong></td>
<td>Specifies or retrieves the index of the current preset.</td>
</tr>
<tr>
<td><strong>currentPresetTitle</strong></td>
<td>Retrieves the title of the current preset.</td>
</tr>
<tr>
<td><strong>currentSpeakerName</strong></td>
<td>Retrieves the name of the current speaker setting.</td>
</tr>
<tr>
<td><strong>enableSplineTension</strong></td>
<td>Specifies or retrieves a value indicating whether spline tension is enabled or disabled.</td>
</tr>
<tr>
<td><strong>enhancedAudio</strong></td>
<td>Specifies or retrieves a value indicating whether enhanced audio is turned on.</td>
</tr>
<tr>
<td><strong>gainLevel1</strong></td>
<td>Specifies or retrieves the gain level of band 1.</td>
</tr>
<tr>
<td><strong>gainLevel2</strong></td>
<td>Specifies or retrieves the gain level of band 2.</td>
</tr>
<tr>
<td><strong>gainLevel3</strong></td>
<td>Specifies or retrieves the gain level of band 3.</td>
</tr>
<tr>
<td><strong>gainLevel4</strong></td>
<td>Specifies or retrieves the gain level of band 4.</td>
</tr>
<tr>
<td><strong>gainLevel5</strong></td>
<td>Specifies or retrieves the gain level of band 5.</td>
</tr>
<tr>
<td><strong>gainLevel6</strong></td>
<td>Specifies or retrieves the gain level of band 6.</td>
</tr>
<tr>
<td><strong>gainLevel7</strong></td>
<td>Specifies or retrieves the gain level of band 7.</td>
</tr>
<tr>
<td><strong>gainLevel8</strong></td>
<td>Specifies or retrieves the gain level of band 8.</td>
</tr>
<tr>
<td><strong>gainLevel9</strong></td>
<td>Specifies or retrieves the gain level of band 9.</td>
</tr>
<tr>
<td><strong>gainLevel10</strong></td>
<td>Specifies or retrieves the gain level of band 10.</td>
</tr>
<tr>
<td><strong>normalization</strong></td>
<td>Specifies or retrieves a value indicating whether normalization is enabled.</td>
</tr>
<tr>
<td><strong>normalizationAverage</strong></td>
<td>Retrieves the average normalization value.</td>
</tr>
<tr>
<td><strong>normalizationPeak</strong></td>
<td>Retrieves the peak normalization value.</td>
</tr>
<tr>
<td><strong>presetCount</strong></td>
<td>Retrieves the number of presets available.</td>
</tr>
<tr>
<td><strong>speakerSize</strong></td>
<td>Specifies or retrieves the index of the current speaker size.</td>
</tr>
</tbody>
</table>
The **EQUALIZERSETTINGS** element supports the following methods.

### Method

**nextPreset**
Applies the next equalizer preset.

**presetTitle**
Retrieves the name of the equalizer preset with the specified index.

**previousPreset**
Applies the previous equalizer preset.

**reset**
Resets the gain levels of all bands to zero decibels.

The **EQUALIZERSETTINGS** element can implement the `attribute_onchange` event handlers.

See Also

- [Skin Programming Reference](#)

--

The **EQUALIZERSETTINGS** element can implement the `attribute_onchange` event handlers.

### bands

The **bands** attribute retrieves the number of frequency bands supported.

#### Syntax

```
elementID.bands
```

#### Possible Values

This attribute is a read-only **Number (long)**.

#### Requirements

Windows Media Player version 7.0 or later.
The `bypass` attribute specifies or retrieves a value indicating whether the equalizer filter is bypassed in the filter graph.

**Syntax**

```
elementID.bypass
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The equalizer filter is bypassed.</td>
</tr>
<tr>
<td>false</td>
<td>The equalizer filter is used.</td>
</tr>
</tbody>
</table>

**Remarks**

If this attribute is not specified, the previous value will be retained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- EQUALIZERSETTINGS Element
EQUALIZERSETTINGS.crossFade

The `crossFade` attribute specifies or retrieves a value indicating whether cross-fade is enabled.

Syntax

```
    elementID.crossFade
```

Possible Values

This attribute is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Cross-fade is enabled.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Cross-fade is disabled.</td>
</tr>
</tbody>
</table>

Remarks

Cross-fade is an audio processing feature that gradually decreases the volume of one media item near the end of its playback while simultaneously starting playback of the next media item at minimum volume and gradually increasing it to normal volume. The overlap between the start of the second media item and the end of the first media item is specified by the `crossFadeWindow` attribute.

Requirements

Windows Media Player 9 Series or later.

See Also

- `EQUALIZERSETTINGS` Element
- `EQUALIZERSETTINGS.crossFadeWindow`
EQUALIZERSETTINGS.crossFadeWindow

The `crossFadeWindow` attribute specifies or retrieves the amount of cross-fade overlap in milliseconds.

Syntax

```
elementID.crossFadeWindow
```

Possible Values

This attribute is a read/write `Number (long)` ranging from 0 to 10,000 with a default value of 250.

Requirements

Windows Media Player 9 Series or later.

See Also

- `EQUALIZERSETTINGS.Element`
- `EQUALIZERSETTINGS.crossFade`

EQUALIZERSETTINGS.currentPreset

The `currentPreset` attribute specifies or retrieves the index of the current preset.

Syntax

```
elementID.currentPreset
```

Possible Values

This attribute is a read/write `Number (long)`.

Remarks

If this attribute is not specified, the previous value will be retained.
Requirements

Windows Media Player version 7.0 or later.

See Also

- **EQUALIZERSETTINGS Element**

EQUALIZERSETTINGS.currentPresetTitle

The `currentPresetTitle` attribute retrieves the title of the current preset.

Syntax

```
elementID.currentPresetTitle
```

Possible Values

This attribute is a read-only `String`.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **EQUALIZERSETTINGS Element**
**EQUALIZERSETTINGS.currentSpeakerName**

The `currentSpeakerName` attribute retrieves the name of the current speaker setting.

**Syntax**

```
elementID.currentSpeakerName
```

**Possible Values**

This attribute is a read-only `String` containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Headphones</td>
<td>The current speakers are headphones.</td>
</tr>
<tr>
<td>Normal Speakers</td>
<td>The current speakers are of normal size.</td>
</tr>
<tr>
<td>Large Speakers</td>
<td>The current speakers are large.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- EQUALIZERSETTINGS.Element
- EQUALIZERSETTINGS.speakerSize

---

**EQUALIZERSETTINGS.enableSplineTension**

The `enableSplineTension` attribute specifies or retrieves a value indicating whether spline tension is enabled.

**Syntax**

```
elementID.enableSplineTension
```

**Possible Values**

---

© 2000-2003 Microsoft Corporation. All rights reserved.
This attribute is a read/write **Boolean** with a default value of true.

**Remarks**

Enabling spline tension allows the user to adjust the equalizer bandwidths more smoothly, so that there are no large jumps between the frequency levels.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- **EQUALIZERSETTINGS** Element
- **EQUALIZERSETTINGS.splineTension**

---

**EQUALIZERSETTINGS.enhancedAudio**

The **enhancedAudio** attribute specifies or retrieves a value indicating whether enhanced audio is turned on.

**Syntax**

\[\text{elementID}.\text{enhancedAudio}\]

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Enhanced audio is turned on.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Enhanced audio is turned off.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player 9 Series or later.

**See Also**
EQUALIZERSETTINGS.gainLevels

The **gainLevels** attribute specifies or retrieves the gain level of the band corresponding to the index provided.

**Syntax**

```plaintext
elementID.gainLevels(theBand)
```

**Parameters**

- **theBand**

  **Number (long)** between 1 and 10 indicating the index of the band.

**Possible Values**

This attribute is a read/write **Number (float)** with a value normally ranging from –20 to +20.

**Remarks**

This attribute takes a parameter, but its value is specified in script code the same way as other attribute values. It cannot be specified in the EQUALIZERSETTINGS element, nor can it be used with the `wmpprop` listening attribute. Instead, the numbered gain level attributes are provided for these situations.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **EQUALIZERSETTINGS Element**
- **Listening Attributes**
EQUALIZERSETTINGS.gainLevel1

The gainLevel1 attribute specifies or retrieves the gain level of band 1.

Syntax

elementID.gainLevel1

Possible Values

This attribute is a read/write Number (float) with a value normally ranging from –20 to +20. It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 31Hz.

If this attribute is not specified, the previous value will be retained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.gainLevels
EQUALIZERSETTINGS.gainLevel2

The **gainLevel2** attribute specifies or retrieves the gain level of band 2.

**Syntax**

```
$elementID.gainLevel2
```

**Possible Values**

This attribute is a read/write **Number (float)** with a value normally ranging from –20 to +20. It has a default value of zero.

**Remarks**

This attribute adjusts the portion of the audio frequency spectrum centered on 62Hz.

If this attribute is not specified, the previous value will be retained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [EQUALIZERSETTINGS Element](#)
- [EQUALIZERSETTINGS.gainLevels](#)

---

EQUALIZERSETTINGS.gainLevel3

The **gainLevel3** attribute specifies or retrieves the gain level of band 3.

**Syntax**

```
$elementID.gainLevel3
```

**Possible Values**
This attribute is a read/write Number (float) with a value normally ranging from –20 to +20. It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 125Hz.

If this attribute is not specified, the previous value will be retained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.gainLevels

EQUALIZERSETTINGS.gainLevel4

The gainLevel4 attribute specifies or retrieves the gain level of band 4.

Syntax

elementID.gainLevel4

Possible Values

This attribute is a read/write Number (float) with a value normally ranging from –20 to +20. It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 250Hz.

If this attribute is not specified, the previous value will be retained.

Requirements
EQUALIZERSETTINGS.gainLevel5

The `gainLevel5` attribute specifies or retrieves the gain level of band 5.

Syntax

```
elementID.gainLevel5
```

Possible Values

This attribute is a read/write `Number (float)` with a value normally ranging from –20 to +20. It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 500Hz.

If this attribute is not specified, the previous value will be retained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.gainLevels
EQUALIZERSETTINGS.gainLevel6

The `gainLevel6` attribute specifies or retrieves the gain level of band 6. It has a default value of zero.

Syntax

```
elementID.gainLevel6
```

Possible Values

This attribute is a read/write `Number (float)` with a value normally ranging from –20 to +20. It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 1kHz.

If this attribute is not specified, the previous value will be retained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.gainLevels

EQUALIZERSETTINGS.gainLevel7

The `gainLevel7` attribute specifies or retrieves the gain level of band 7.
Syntax

elementID.gainLevel7

Possible Values

This attribute is a read/write Number (float) with a value normally ranging from \(-20\) to \(+20\). It has a default value of zero.

Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 2kHz.

If this attribute is not specified, the previous value will be retained.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.gainLevels

EQUALIZERSETTINGS.gainLevel8

The gainLevel8 attribute specifies or retrieves the gain level of band 8.

Syntax

elementID.gainLevel8

Possible Values

This attribute is a read/write Number (float) with a value normally ranging from \(-20\) to \(+20\). It has a default value of zero.

Remarks
This attribute adjusts the portion of the audio frequency spectrum centered on 4kHz.

If this attribute is not specified, the previous value will be retained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `EQUALIZERSETTINGS Element`
- `EQUALIZERSETTINGS.gainLevels`

---

**EQUALIZERSETTINGS.gainLevel9**

The `gainLevel9` attribute specifies or retrieves the gain level of band 9.

**Syntax**

`elementID.gainLevel9`

**Possible Values**

This attribute is a read/write `Number (float)` with a value normally ranging from –20 to +20. It has a default value of zero.

**Remarks**

This attribute adjusts the portion of the audio frequency spectrum centered on 8kHz.

If this attribute is not specified, the previous value will be retained.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `EQUALIZERSETTINGS Element`
The `gainLevel10` attribute specifies or retrieves the gain level of band 10.

### Syntax

```
elementID.gainLevel10
```

### Possible Values

This attribute is a read/write `Number (float)` with a value normally ranging from –20 to +20. It has a default value of zero.

### Remarks

This attribute adjusts the portion of the audio frequency spectrum centered on 16kHz.

If this attribute is not specified, the previous value will be retained.

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- [EQUALIZERSETTINGS Element](#)
- [EQUALIZERSETTINGS.gainLevels](#)
EQUALIZERSETTINGS.nextPreset

The `nextPreset` method applies the next equalizer preset.

Syntax

elementID.nextPreset()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Example

```xml
<SUBVIEW id="eqtray">
    <EQUALIZERSETTINGS id="eq"/>
    <BUTTON
        id="nextPreset"
        onClick="eq.nextPreset();"
    />
    <BUTTON
        id="prevPreset"
        onClick="eq.previousPreset();"
    />
    <TEXT
        id="currentPreset"
        value="wmpprop:eq.currentPresetTitle"
    />
</SUBVIEW>
```

Remarks

If the current preset is the last one available, the first preset is made current.

Requirements

Windows Media Player version 7.0 or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.previousPreset
EQUALIZERSETTINGS.normalization

The normalization attribute specifies or retrieves a value indicating whether normalization is enabled.

Syntax

`elementID.normalization`

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Normalization is enabled.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Normalization is disabled.</td>
</tr>
</tbody>
</table>

Remarks

When normalization is enabled, the audio signal for an entire digital media file is scaled up to the maximum value. This allows multiple files to be played at approximately the same volume without requiring volume adjustments.

Requirements

Windows Media Player 9 Series or later.

See Also

- [EQUALIZERSETTINGS Element](#)
- [EQUALIZERSETTINGS.normalizationAverage](#)
- [EQUALIZERSETTINGS.normalizationPeak](#)
EQUALIZERSETTINGS.normalizationAverage

The **normalizationAverage** attribute retrieves the average normalization value.

**Syntax**

```
elementID.normalizationAverage
```

**Possible Values**

This attribute is a read-only **Number (float)**.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [EQUALIZERSETTINGS Element](#)
- [EQUALIZERSETTINGS.normalization](#)
- [EQUALIZERSETTINGS.normalizationPeak](#)

EQUALIZERSETTINGS.normalizationPeak

The **normalizationPeak** attribute retrieves the peak normalization value.

**Syntax**

```
elementID.normalizationPeak
```

**Possible Values**
This attribute is a read-only **Number (float)**.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- **EQUALIZERSETTINGS Element**
- **EQUALIZERSETTINGS.normalization**
- **EQUALIZERSETTINGS.normalizationAverage**

---

**EQUALIZERSETTINGS.presetCount**

The **presetCount** attribute retrieves the number of presets available.

**Syntax**

`elementID.presetCount`

**Possible Values**

This attribute is a read-only **Number (long)**.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **EQUALIZERSETTINGS Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.
EQUALIZERSETTINGS.presetTitle

The `presetTitle` method retrieves the name of the equalizer preset with the specified index.

**Syntax**

```plaintext
elementID.presetTitle(index)
```

**Parameters**

`index`

Number (long) containing the preset index.

**Return Values**

This method returns a `String`.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- EQUALIZERSETTINGS Element

EQUALIZERSETTINGS.previousPreset

The `previousPreset` method applies the previous equalizer preset.

**Syntax**

```plaintext
elementID.previousPreset()
```
Removal of a preset changes the current preset.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `EQUALIZERSETTINGS.Element`
- `EQUALIZERSETTINGS.nextPreset`

© 2000-2003 Microsoft Corporation. All rights reserved.

EQUALIZERSETTINGS.reset

The `reset` method resets the gain levels of all bands to zero decibels.

Syntax

```
elementID.reset()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements
EQUALIZERSETTINGS.speakerSize

The `speakerSize` attribute specifies or retrieves the index number of the current speaker size.

Syntax

```
elementID.speakerSize
```

Possible Values

This attribute is a read/write `Number (long)` containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The current speakers are headphones.</td>
</tr>
<tr>
<td>1</td>
<td>The current speakers are of normal size.</td>
</tr>
<tr>
<td>2</td>
<td>The current speakers are large.</td>
</tr>
</tbody>
</table>

Remarks

The friendly name of the speaker size can be retrieved using the `currentSpeakerName` attribute.

This attribute is ignored if `enhancedAudio` is set to false.

Requirements

Windows Media Player 9 Series or later.

See Also

- EQUALIZERSETTINGS Element
- EQUALIZERSETTINGS.currentSpeakerName
The `splineTension` attribute specifies or retrieves the spline tension for the equalizer control.

Syntax

```
elementID.splineTension
```

Possible Values

This attribute is a read/write `Number (float)` ranging from 0.0 (no tension) to 10.0 (full tension) with a default value of 3.0.

Requirements

Windows Media Player 9 Series or later.

See Also

- `EQUALIZERSETTINGS.Element`
- `EQUALIZERSETTINGS.enhancedAudio`
- `EQUALIZERSETTINGS.enableSplineTension`

The `truBassLevel` attribute specifies or retrieves the SRS TruBass level.
**Syntax**

`elementID.truBassLevel`

**Possible Values**

This attribute is a read/write `Number (long)` ranging from 0 to 100 with a default value of 50.

**Remarks**

TruBass is an effect that enhances the sound of the bass levels of the audio track. This attribute is ignored if `enhancedAudio` is set to false.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `EQUALIZERSETTINGS.Element`
- `EQUALIZERSETTINGS.enhancedAudio`

**EQUALIZERSETTINGS.wowLevel**

The `wowLevel` attribute specifies or retrieves the SRS WOW Effect level.

**Syntax**

`elementID.wowLevel`

**Possible Values**

This attribute is a read/write `Number (long)` ranging from 0 to 100 with a default value of 50.

**Remarks**

The SRS WOW Effect is an audio enhancement effect. This attribute is ignored if `enhancedAudio` is set to false.
ITEM Element

The ITEM element represents an item in a list box or pop-up controls. ITEM elements are always children of LISTBOX or POPUP elements.

The ITEM element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies or retrieves the text that displays for a list box or pop-up item.</td>
</tr>
</tbody>
</table>

Note  This element requires Windows Media Player for Windows XP or later.

See Also

- Skin Programming Reference

© 2000-2003 Microsoft Corporation. All rights reserved.
ITEM.value

The value attribute specifies or retrieves the text that displays for a list box or pop-up item.

Syntax

```
elementID.value
```

Possible Values

This attribute is a read/write String.

Requirements

Windows Media Player for Windows XP or later.

See Also

- ITEM Element

LISTBOX Element

The LISTBOX element provides a way for users to select items from a list. These items can be specified by using ITEM elements as children of the LISTBOX element. If you need a list box control that is displayed only when needed, use the POPUP element, which is identical to the LISTBOX element except for the default value of the popUp attribute, which dictates its display behavior.

The LISTBOX element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Specifies or retrieves the background color in the list box control.</td>
</tr>
<tr>
<td>border</td>
<td>Specifies or retrieves a value indicating whether the list box control has a border. Can only be set at design time.</td>
</tr>
<tr>
<td>firstVisibleItem</td>
<td>Specifies or retrieves the index of the first visible line</td>
</tr>
</tbody>
</table>
The **LISTBOX** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>focusItem</td>
<td>Specifies or retrieves the line that contains focus.</td>
</tr>
<tr>
<td>fontFace</td>
<td>Specifies or retrieves the font for the list box control.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Specifies or retrieves the font size for the list box control.</td>
</tr>
<tr>
<td>fontStyle</td>
<td>Specifies or retrieves the font style for the list box control.</td>
</tr>
<tr>
<td>foregroundColor</td>
<td>Specifies or retrieves the text color in the list box control.</td>
</tr>
<tr>
<td>itemCount</td>
<td>Retrieves the number of items in the list box control.</td>
</tr>
<tr>
<td>multiSelect</td>
<td>Specifies or retrieves a value indicating whether the user can select multiple lines. Can only be set at design time.</td>
</tr>
<tr>
<td>popUp</td>
<td>Specifies a value indicating whether the element represents a popup or list box control.</td>
</tr>
<tr>
<td>readOnly</td>
<td>Specifies or retrieves a value indicating whether text is read-only or text can be selected by the user.</td>
</tr>
<tr>
<td>selectedItem</td>
<td>Specifies or retrieves the index of the item selected in the list box control.</td>
</tr>
<tr>
<td>sorted</td>
<td>Specifies or retrieves a value indicating whether the list box control is sorted alphabetically.</td>
</tr>
</tbody>
</table>

The **LISTBOX** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendItem</td>
<td>Inserts new text at the end of the list box control.</td>
</tr>
<tr>
<td>deleteAll</td>
<td>Deletes all items from the list box control.</td>
</tr>
<tr>
<td>deleteItem</td>
<td>Deletes the list box control item at the specified index.</td>
</tr>
<tr>
<td>dismiss</td>
<td>Hides the control.</td>
</tr>
<tr>
<td>findItem</td>
<td>Searches for a given string starting with the item following the specified item index.</td>
</tr>
<tr>
<td>getItem</td>
<td>Retrieves the text for the item with the specified index.</td>
</tr>
<tr>
<td>getNextSelectedItem</td>
<td>Retrieves the next selected item in the list box control starting at the item after the one with the specified index.</td>
</tr>
<tr>
<td>insertItem</td>
<td>Inserts the specified text at the specified index in the list box control.</td>
</tr>
</tbody>
</table>
LISTBOX.appendItem

The `appendItem` method inserts new text at the end of the list box control.

**Syntax**

```
appendItem(elementID, newVal)
```

**Parameters**

- `newVal`  
  
  **String** containing the text to be inserted.

**Return Values**

This method does not return a value.

**Requirements**

Windows Media Player for Windows XP or later.

See Also

- [LISTBOX Element](#)
LISTBOX.backgroundColor

The `backgroundColor` attribute specifies or retrieves the background color in the list box control.

**Syntax**

```
elementID.backgroundColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value or the value "none". It has a default value equal to the Windows background color.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- Color Reference
- LISTBOX Element
- LISTBOX.foregroundColor

LISTBOX.border

The `border` attribute specifies or retrieves a value indicating whether the list box control has a border.
Syntax

elementID.border

Possible Values

This attribute is a **Boolean** value.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The list box control has a border.</td>
</tr>
<tr>
<td>false</td>
<td>The list box control does not have a border.</td>
</tr>
</tbody>
</table>

Remarks

The border appears as a two-pixel-wide sunken border around the list box control with system colors.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element

LISTBOX.deleteAll

The **deleteAll** method deletes all items from the list box control.

Syntax

elementID.deleteAll()  

Parameters

This method takes no parameters.

Return Values
This method does not return a value.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element

LISTBOX.deleteItem

The `deleteItem` method deletes the list box control item at the specified index.

Syntax

```
elementID.deleteItem(index)
```

Parameters

- `index`
  
  *Number* (*long*) containing the index of the item to delete.

Return Values

This method does not return a value.

Remarks

The lines below the deleted line will move up one index value.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element
LISTBOX.dismiss

The `dismiss` method hides the control.

**Syntax**

```
elementID.dismiss()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

This method is primarily used with the `POPUP` element, which represents a list box control that is displayed only when needed.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- `LISTBOX Element`
- `LISTBOX.show`
- `POPUP Element`
LISTBOX.findItem

The **findItem** method searches for a given string starting with the item following the specified item index.

**Syntax**

```
elementID.findItem(startIndex, searchString)
```

**Parameters**

- **startIndex**
  
  *Number (long)* containing the index of the item at which to start the search.

- **searchString**
  
  *String* containing the text to search for.

**Return Values**

This method returns a *Number (long)* containing the index of the item that contains the string.

**Remarks**

To start the search at the first line of the list box control, use –1 as the **startIndex**. To continue to search for text after the first line is found, use the returned line index as the **startIndex**, and the search will start at the next line. This method will search for substrings and is not case-sensitive.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- [LISTBOX Element](#)
LISTBOX.firstVisibleItem

The firstVisibleItem attribute specifies or retrieves the index of the first visible line in the list box control.

Syntax

```
elementID.firstVisibleItem
```

Possible Values

This attribute is a read/write Number (long).

Remarks

This attribute is useful for determining the first visible line when the list box is scrolled.

Requirements

Windows Media Player for Windows XP or later.

See Also

- [LISTBOX Element](#)

LISTBOX.focusItem

The focusItem attribute specifies or retrieves the line that contains focus.

Syntax

```
elementID.focusItem
```

Possible Values
This attribute is a read/write **Number (long)**.

**Remarks**

The focus item and selected item could be different. An item can be selected through script, but another item can be given focus through script. The item with focus will have a dotted rectangle around it, and will be selected if the ENTER key is pressed.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- [LISTBOX Element](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**LISTBOX.fontFace**

The **fontFace** attribute specifies or retrieves the font for the list box control.

**Syntax**

`elementID.fontFace`

**Possible Values**

This attribute is a read/write **String**.

**Remarks**

This attribute can be the name of any valid font available in Windows. Windows Media Player will not support installing fonts, so choose a font that you know will be on the intended system.

If the **fontFace** specified is not available on the user's system, the list box control defaults to the Windows system font. The default value for English-language systems is "Tahoma". For international systems, the default is loaded from a resource file.

**Requirements**

---
LISTBOX.fontSize

The fontSize attribute specifies or retrieves the font size for the list box control.

Syntax

elementID.fontSize

Possible Values

This attribute is a read/write Number (long) specifying the font size in points. The default value is 10.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element
- LISTBOX(fontSize
- LISTBOX.fontStyle
LISTBOX.fontStyle

The fontStyle attribute specifies or retrieves the font style for the list box control.

Syntax

```
elementID.fontStyle
```

Possible Values

This attribute is a read/write String containing one or more of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold font style.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic font style.</td>
</tr>
<tr>
<td>Underline</td>
<td>Underline font style.</td>
</tr>
<tr>
<td>Strikeout</td>
<td>Strikeout font style.</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal font style.</td>
</tr>
</tbody>
</table>

Remarks

Any combination of the values can be used, separated with spaces. The Normal style has priority over all other values, and any others specified along with Normal will be ignored.

For rendering purposes, Normal is the default font style. The default value retrieved from this attribute is "" (empty string).

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element
- LISTBOX(fontSize)
- LISTBOX.fontStyle
LISTBOX.foregroundColor

The **foregroundColor** attribute specifies or retrieves the text color in the list box control.

**Syntax**

```
elementID.foregroundColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value equal to the Windows text color.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- [Color Reference](#)
- LISTBOX Element
- LISTBOX.backgroundColor

LISTBOX.getItem

The **getItem** method retrieves the text for the item with the specified index.

**Syntax**

```
elementID.getItem(index)
```

**Parameters**
**index**

**Number (long)** containing the index of the line to retrieve.

**Return Values**

This method returns a **String**.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- **LISTBOX Element**

---

**LISTBOX.getNextSelectedItem**

The **getNextSelectedItem** method retrieves the next selected item in the list box control starting at the item after the one with the specified index.

**Syntax**

```plaintext
elementID.getNextSelectedItem(startIndex)
```

**Parameters**

- **startIndex**

  **Number (long)** containing the index of the item that precedes the item being retrieved.

**Return Values**

This method returns a **Number (long)** containing the index of the next selected item.

**Remarks**

To start search from the beginning, use –1 for the start index.
LISTBOX.insertItem

The **insertItem** method inserts the specified text at the specified index in the list box control.

**Syntax**

```
elementID.insertItem(index, text)
```

**Parameters**

- **index**
  
  Number (**long**) containing the index of the line to retrieve.

- **text**
  
  String containing the text to be inserted.

**Return Values**

This method does not return a value.

**Remarks**

When a line is inserted, the line indexes for the lines below the inserted line increase by one.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- **LISTBOX Element**
LISTBOX.itemCount

The **itemCount** attribute retrieves the number of items in the list box control.

**Syntax**

```plaintext
elementID.itemCount
```

**Possible Values**

This attribute is a read-only **Number (long)**.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- **LISTBOX Element**

---

LISTBOX.multiSelect

The **multiSelect** attribute specifies or retrieves a value indicating whether the user can select multiple lines. Can only be set at design time.
Syntax

elementID.multiSelect

Possible Values

This attribute is a **Boolean** specified at design time and read-only thereafter.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The user can select multiple lines.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The user can only select individual lines.</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player for Windows XP or later.

See Also

- [LISTBOX Element](#)

---

LISTBOX.popUp

The **popUp** attribute specifies a value indicating whether the element represents a popup or list box control.

Syntax

```
<ELEMENT popUp="value"/>
```

Possible Values

This attribute is a **Boolean** specified at design time only.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The element represents a popup control.</td>
</tr>
<tr>
<td>false</td>
<td>The element represents a list box control.</td>
</tr>
</tbody>
</table>
Remarks

The **POPUP** element represents a list box control that is displayed only when needed. It is identical to the **LISTBOX** element except for the default value of this attribute, which changes the display behavior. For **LISTBOX** elements, the default value is false. For **POPUP** elements, the default value is true. Instead of specifying this attribute, the **LISTBOX** or **POPUP** element should be used to according to the desired behavior.

Requirements

Windows Media Player for Windows XP or later.

See Also

- **LISTBOX Element**
- **POPUP Element**

© 2000-2003 Microsoft Corporation. All rights reserved.

LISTBOX.readOnly

The **readOnly** attribute specifies or retrieves a value indicating whether text is read only or can be selected by the user. Can only be set at design time.

Syntax

```xml
elementID.readOnly
```

Possible Values

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Text is read only.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Text can be selected by the user.</td>
</tr>
</tbody>
</table>

Requirements
LISTBOX.replaceItem

The **replaceItem** method replaces the text at the specified index with the specified text.

**Syntax**

```
elementType.replaceItem(index, text)
```

**Parameters**

- **index**
  
  **Number** (**long**) containing the index of the line to retrieve.

- **text**
  
  **String** containing the new text.

**Return Values**

This method does not return a value.

**Requirements**

Windows Media Player for Windows XP or later.

See Also

- **LISTBOX Element**
LISTBOX.selectedItem

The `selectedItem` attribute specifies or retrieves the index of the item selected in the list box control.

**Syntax**

```
elementID.selectedItem
```

**Possible Values**

This attribute is a read/write `Number (long)`.

**Remarks**

This property will select one line in the list box control. Any other selected line will be unselected.

**Requirements**

Windows Media Player for Windows XP or later.

**See Also**

- [LISTBOX Element](#)

---

LISTBOX.setSelectedState

The `setSelectedState` method selects or unselects the item with the specified index.

**Syntax**

```
elementID.setSelectedState(index)
```
elementID.setSelectedState(index, selected)

Parameters

index

Number (long) containing the index of the item to select or unselect.

selected

Boolean value indicating whether the item is to be selected (true) or unselected (false).

Return Values

This method does not return a value.

Remarks

This is used to select or unselect multiple lines.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element

LISTBOX.show

The show method displays the control.

Syntax

elementID.show()
Return Values

This method does not return a value.

Remarks

This method is primarily used with the POPUP element, which represents a list box control that is displayed only when needed.

Requirements

Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element
- LISTBOX.dismiss
- POPUP Element

LISTBOX.sorted

The sorted attribute specifies or retrieves a value indicating whether the list box control is sorted alphabetically. Can only be set at design time.

Syntax

```
elementID.sorted
```

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The list box control is sorted alphabetically.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The list box control is not sorted alphabetically.</td>
</tr>
</tbody>
</table>
PLAYER Element

The PLAYER element lets you define event handlers and specify the URL property of the Player object at design time within a skin definition file. Within script code, the Player object is accessed through the player global attribute rather than through a name specified by an id attribute, which is not supported by the PLAYER element.

The PLAYER element supports the following attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>Specifies or retrieves the name of the file to play.</td>
</tr>
</tbody>
</table>

The PLAYER element can implement event handlers for the following events fired from the Player object.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AudioLanguageChange</td>
<td>Occurs when the current audio language changes.</td>
</tr>
<tr>
<td>Buffering</td>
<td>Occurs when the player begins or ends buffering.</td>
</tr>
<tr>
<td>CdromMediaChange</td>
<td>Occurs when the CD-ROM media changes.</td>
</tr>
<tr>
<td>CurrentItemChange</td>
<td>Occurs when the current item changes.</td>
</tr>
<tr>
<td>CurrentMediaItemAvailable</td>
<td>Occurs when the current media item becomes available.</td>
</tr>
<tr>
<td>CurrentPlaylistChange</td>
<td>Occurs when the current playlist changes.</td>
</tr>
<tr>
<td>CurrentPlaylistItemAvailable</td>
<td>Occurs when the current playlist item becomes available.</td>
</tr>
</tbody>
</table>
**DomainChange**
- Occurs when the DVD domain changes.

**Error**
- Occurs when the Windows Media Player control has an error condition.

**MarkerHit**
- Occurs when the player encounters a marker in the clip.

**MediaChange**
- Occurs when a media item changes.

**MediaCollectionAttributeStringAdded**
- Occurs when an attribute value is added to Media Library.

**MediaCollectionAttributeStringChanged**
- Occurs when an attribute value in Media Library is changed.

**MediaCollectionAttributeStringRemoved**
- Occurs when an attribute value is removed from Media Library.

**MediaCollectionChange**
- Occurs when the MediaCollection object changes.

**MediaError**
- Occurs when the media object has an error condition.

**ModeChange**
- Occurs when switching between shuffle and normal mode.

**OpenPlaylistSwitch**
- Occurs when a title on a DVD begins playing.

**OpenStateChange**
- Occurs when `player.openState` changes.

**PlayerDockedStateChange**
- Occurs when a remoted Windows Media Player control docks or undocks.

**PlaylistChange**
- Occurs when a playlist changes.

**PlaylistCollectionChange**
- Occurs when something changes in the playlist collection.

**PlaylistCollectionPlaylistAdded**
- Occurs when a playlist is added to the playlist collection.

**PlaylistCollectionPlaylistRemoved**
- Occurs when a playlist is removed from the playlist collection.

**PlayStateChange**
- Occurs when `player.playState` changes.

**PositionChange**
- Occurs when `player.controls.currentPosition` changes.

**ScriptCommand**
- Occurs when the player encounters a script command embedded in the Windows Media file.

**StatusChange**
- Occurs when the `status` property changes value.

**SwitchedToControl**
- Occurs when a remoted Windows Media Player control switches to the docked state.

**SwitchedToPlayerApplication**
- Occurs when a remoted Windows Media Player
The PLAYLIST element provides a way to organize media items in a list for easy manipulation using the following attributes and methods. Predefined PLAYLIST elements are also provided for convenience. Customized columns can be specified for a playlist by including COLUMN elements as children of the PLAYLIST element.

The PLAYLIST element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowColumnSorting</td>
<td>Specifies or retrieves a value indicating whether sorting on column headers is allowed.</td>
</tr>
<tr>
<td>allowItemEditing</td>
<td>Specifies or retrieves a value indicating whether items in a playlist support in-place editing.</td>
</tr>
<tr>
<td>backgroundColor</td>
<td>Specifies or retrieves the background color.</td>
</tr>
<tr>
<td>backgroundImage</td>
<td>Specifies or retrieves the background image.</td>
</tr>
<tr>
<td>checkboxesVisible</td>
<td>Specifies or retrieves a value indicating whether checkboxes are visible.</td>
</tr>
<tr>
<td>columnCount</td>
<td>Retrieves the number of columns shown.</td>
</tr>
<tr>
<td>columnOrder</td>
<td>Specifies or retrieves the order of the playlist columns.</td>
</tr>
<tr>
<td>columns</td>
<td>Defines the columns that appear in the Playlist control.</td>
</tr>
<tr>
<td>columnsVisible</td>
<td>Specifies or retrieves a value indicating whether columns are shown.</td>
</tr>
<tr>
<td>copying</td>
<td>Retrieves a value indicating whether the Playlist control is in the act of copying.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>disabledItemColor</td>
<td>Specifies or retrieves the color of a disabled CD track or of online content when offline.</td>
</tr>
<tr>
<td>dropDownBackgroundImage</td>
<td>Specifies or retrieves the name of the image that displays in the background of the drop-down list.</td>
</tr>
<tr>
<td>dropDownImage</td>
<td>Specifies or retrieves the name of the image used for the drop-down list button that is displayed at the right edge of the drop-down list.</td>
</tr>
<tr>
<td>dropDownList</td>
<td>Specifies or retrieves a value indicating which elements show up in the drop-down list for a given instance of the Playlist control.</td>
</tr>
<tr>
<td>dropDownToolTip</td>
<td>Specifies or retrieves the ToolTip shown when the user hovers over the Playlist control drop-down menu.</td>
</tr>
<tr>
<td>dropDownVisible</td>
<td>Specifies or retrieves a value indicating whether the Playlist drop-down selector is visible.</td>
</tr>
<tr>
<td>editButtonVisible</td>
<td>Specifies or retrieves a value indicating whether the Playlist edit button is visible.</td>
</tr>
<tr>
<td>foregroundColor</td>
<td>Specifies or retrieves the foreground color.</td>
</tr>
<tr>
<td>hueShift</td>
<td>Specifies or retrieves the amount by which the hue of the drop-down images is shifted.</td>
</tr>
<tr>
<td>itemCount</td>
<td>Retrieves the number of items currently displayed in the Playlist control.</td>
</tr>
<tr>
<td>itemErrorColor</td>
<td>Specifies or retrieves the highlight color that indicates a playlist item that has an error condition.</td>
</tr>
<tr>
<td>itemMedia</td>
<td>Retrieves the Media object corresponding to the given index in the playlist control.</td>
</tr>
<tr>
<td>itemPlayingBackgroundColor</td>
<td>Specifies or retrieves the background color of the currently playing Playlist item.</td>
</tr>
<tr>
<td>itemPlayingColor</td>
<td>Specifies or retrieves the highlight color that indicates the currently playing item in the playlist.</td>
</tr>
<tr>
<td>itemPlaylist</td>
<td>Retrieves the playlist for the media item that is displayed at the given index in the playlist control.</td>
</tr>
<tr>
<td>leftStatus</td>
<td>Specifies or retrieves the status text that is displayed on the left side and bottom of the Playlist control.</td>
</tr>
<tr>
<td>playlist</td>
<td>Specifies or retrieves the Playlist object that the Playlist control provides an interface to.</td>
</tr>
<tr>
<td>playlistItemsVisible</td>
<td>Specifies or retrieves a value indicating whether items in the playlist are visible.</td>
</tr>
<tr>
<td>rightStatus</td>
<td>Specifies or retrieves the status text that is displayed on the right side and bottom of the Playlist control.</td>
</tr>
<tr>
<td>saturation</td>
<td>Specifies or retrieves the saturation value of the drop-down</td>
</tr>
</tbody>
</table>
statusColor

Specifies or retrieves the color of the status line in the Playlist control.

The **PLAYLIST** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortCopy</td>
<td>Cancels a copy operation.</td>
</tr>
<tr>
<td>addSelectedToPlaylist</td>
<td>Adds the selected item to the playlist.</td>
</tr>
<tr>
<td>copy</td>
<td>Begins a copy operation from the CD.</td>
</tr>
<tr>
<td>deleteSelected</td>
<td>Deletes the selected item from the playlist.</td>
</tr>
<tr>
<td>deleteSelectedFromLibrary</td>
<td>Deletes the selected item from the playlist and from Media Library.</td>
</tr>
<tr>
<td>getNextCheckedItem</td>
<td>Retrieves the index of the next checked item in the playlist following the specified index.</td>
</tr>
<tr>
<td>getNextCheckedItem2</td>
<td>Retrieves the index of the next checked item in the playlist following the specified index. Works with nested playlists.</td>
</tr>
<tr>
<td>getNextSelectedItem</td>
<td>Retrieves the index of the next selected item in the playlist following the specified index.</td>
</tr>
<tr>
<td>getNextSelectedItem2</td>
<td>Retrieves the index of the next selected item in the playlist following the specified index. Works with nested playlists.</td>
</tr>
<tr>
<td>moveSelectedDown</td>
<td>Moves the selected item down one position in the list.</td>
</tr>
<tr>
<td>moveSelectedUp</td>
<td>Moves the selected item up one position in the list.</td>
</tr>
<tr>
<td>setCheckedState</td>
<td>Specifies that the indexed item in the playlist is checked.</td>
</tr>
<tr>
<td>setCheckedState2</td>
<td>Sets the checked state of the item with the specified index in the Playlist control. Works with nested playlists.</td>
</tr>
<tr>
<td>setColumnResizeMode</td>
<td>Specifies how the indexed column sizes itself.</td>
</tr>
<tr>
<td>setColumnWidth</td>
<td>Specifies the column width and changes the resize mode of the column to &quot;wmpcrmFixed&quot;.</td>
</tr>
<tr>
<td>setSelectedState</td>
<td>Specifies that the indexed item in the playlist is selected.</td>
</tr>
<tr>
<td>setSelectedState2</td>
<td>Sets the selected state of the item with the specified index in the Playlist control. Works with nested playlists.</td>
</tr>
<tr>
<td>sortColumn</td>
<td>Sorts the data in the specified column.</td>
</tr>
</tbody>
</table>

The **PLAYLIST** element supports the ambient attributes and can implement the ambient event handlers, except where noted. For more information, see Ambient Attributes and Ambient Event Handlers.

Predefined playlists are normal **PLAYLIST** elements with various common attribute settings specified by default. The following predefined playlists are available.
The `abortCopy` method cancels a copy operation.

**Syntax**

```
elementID.abortCopy()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `PLAYLIST Element`
- `PLAYLIST.copy`
- `PLAYLIST.copying`
PLAYLIST.addSelectedToPlaylist

The `addSelectedToPlaylist` method adds the selected item to a playlist.

Syntax

```
elementID.addSelectedToPlaylist(playlist)
```

Parameters

- `playlist`
  
  `Playlist` object that will receive the selected item.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- [PLAYLIST Element](#)

PLAYLIST.allowColumnSorting

The `allowColumnSorting` attribute specifies or retrieves a value indicating whether sorting column contents is
allowed.

**Syntax**

```
elementID.allowColumnSorting
```

**Possible Values**

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Sorting column contents is allowed.</td>
</tr>
<tr>
<td>false</td>
<td>Sorting column contents is not allowed.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [PLAYLIST Element](#)

---

**PLAYLIST.allowItemEditing**

The `allowItemEditing` attribute specifies or retrieves a value indicating whether items in a playlist will support in-place editing.

**Syntax**

```
elementID.allowItemEditing
```

**Possible Values**

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

© 2000-2003 Microsoft Corporation. All rights reserved.
PLAYLIST.backgroundColor

The `backgroundColor` attribute specifies or retrieves the background color.

**Syntax**

`elementID.backgroundColor`

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value. It has a default value equal to the Windows system window color.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **Color Reference**
- **PLAYLIST Element**
The `backgroundImage` attribute specifies or retrieves the background image.

**Syntax**

`elementID.backgroundImage`

**Possible Values**

This attribute is a read/write `String` containing the name of an image file. It has no default value.

**Remarks**

If the image height and width are smaller than the height and width of the Playlist control, the image is tiled. The supported formats are BMP, JPG, GIF and PNG.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [PLAYLIST Element](#)

---

The `checkboxesVisible` attribute specifies or retrieves a value indicating whether check boxes are visible.

**Syntax**

`elementID.checkboxesVisible`
Possible Values

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Check boxes are visible.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Check boxes are not visible.</td>
</tr>
</tbody>
</table>

Remarks

Check boxes appear in the far left column.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **PLAYLIST Element**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**PLAYLIST.columnCount**

The **columnCount** attribute retrieves the number of columns shown.

Syntax

`elementID.columnCount`

Possible Values

This attribute is a read-only **Number (long)**.

Requirements

Windows Media Player version 7.0 or later.

See Also
The `columnOrder` attribute specifies or retrieves the order of the playlist columns.

**Syntax**

```
$elementID.columnOrder
```

**Possible Values**

This attribute is a read/write `String` specifying a semicolon-delimited list of Playlist column indexes. The default value is "0;1;2;3". Leading and trailing semicolons and spaces should not be present.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `PLAYLIST Element`

---

The `columns` attribute defines the columns that appear in the Playlist control.
Syntax

elementID.columns

Possible Values

This attribute is a read/write String in the following format:

DB_NAME=FRIENDLY_NAME;DB_NAME=FRIENDLY_NAME;

where FRIENDLY_NAME is the value shown in the column header of the Playlist control, and DB_NAME is a value from the following table. Note that a Playlist item might be a media item from Media Library or from a CD, or it might be another Playlist that is either user-created or taken from a CD. Some columns are valid only with certain Playlist items as indicated in the Description column.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Album</td>
<td>The name of the album. Used with media items only.</td>
</tr>
<tr>
<td>Artist</td>
<td>The name of the artist. Not used with user-created Playlists.</td>
</tr>
<tr>
<td>Author</td>
<td>The name of the artist.</td>
</tr>
<tr>
<td>Bitrate</td>
<td>The bit rate of the content. Used only with media items from Media Library.</td>
</tr>
<tr>
<td>CDTrackEnabled</td>
<td>Indicates whether the CD track is enabled. Used only with CD media items.</td>
</tr>
<tr>
<td>Checked</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Copyright</td>
<td>The copyright of the Playlist item. Not used with CD playlists or media items.</td>
</tr>
<tr>
<td>CreationDate</td>
<td>The date and time that the entry in Media Library was created. Used only with items from Media Library.</td>
</tr>
<tr>
<td>DigitallySecure</td>
<td>Indicates whether the item is protected with Windows Media Rights Manager. Used only with media items from Media Library.</td>
</tr>
<tr>
<td>Duration</td>
<td>The duration of the media item.</td>
</tr>
<tr>
<td>FileType</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Genre</td>
<td>The genre of the Playlist item. Not used with playlists from CDs.</td>
</tr>
<tr>
<td>MediaAttribute</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>MediaType</td>
<td>The type of the media item (audio or video).</td>
</tr>
<tr>
<td>MetadataSource</td>
<td>The source of the metadata for this CD track (AMG, WindowsMedia.com, and so on). Used only with CD media items.</td>
</tr>
<tr>
<td>ModifiedBy</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the Playlist item.</td>
</tr>
<tr>
<td>OriginalIndex</td>
<td>If applicable, the corresponding CD track identifier. Used only with CD</td>
</tr>
</tbody>
</table>
Remarks

If one of the columns does not exist in Media Library, it is left blank.

A maximum of 31 columns may be specified.

If you specify a duplicate column, the data in the first column will be left-aligned but all other duplicate columns will be right-aligned. For example, if you have two duration columns, the data will be left-aligned in the first and right-aligned in the second.

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element
- PLAYLIST.columnsVisible

<table>
<thead>
<tr>
<th>Remarks</th>
<th>Media Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlayCount</td>
<td>The number of times the content has been played through to the end. Used only with media items from Media Library.</td>
</tr>
<tr>
<td>PlaylistAttribute</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Rating</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>SourceURL</td>
<td>The path or URL to the content. Used only with media items from Media Library.</td>
</tr>
<tr>
<td>Status</td>
<td>The copying status of a CD track being copied. Used only with CD media items.</td>
</tr>
<tr>
<td>Style</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>TOC</td>
<td>If applicable, the corresponding CD Table of Contents Identifier. Not used with user-created Playlists.</td>
</tr>
</tbody>
</table>
The `columnsVisible` attribute specifies or retrieves a value indicating whether column headers are shown.

**Syntax**

```
elementID.columnsVisible
```

**Possible Values**

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Column headers are shown.</td>
</tr>
<tr>
<td>false</td>
<td>Column headers are not shown.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [PLAYLIST Element](#)
- [PLAYLIST.columns](#)

---

The `copy` method begins a copy operation from the CD.

**Syntax**

```
elementID.copy()
```

**Parameters**

This method takes no parameters.

**Return Values**
This method does not return a value.

Remarks

This method copies only the checked items in the playlist, and works the same way as in the Copy from CD pane in the full mode of the Player. For this method to work, a CD must be in the CD-ROM drive. Set the checkboxesVisible attribute to true to allow users to select individual items on a CD before copying.

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element
- PLAYLIST.abortCopy
- PLAYLIST.copying

© 2000-2003 Microsoft Corporation. All rights reserved.
PLAYLIST.deleteSelected

The `deleteSelected` method deletes the selected item from the playlist.

Syntax

```plaintext
elementID.deleteSelected()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- [PLAYLIST Element](#)
- [PLAYLIST.abortCopy](#)
- [PLAYLIST.copy](#)
PLAYLIST.deleteSelectedFromLibrary

The `deleteSelectedFromLibrary` method deletes the selected item from the playlist and from Media Library.

Syntax

```
elementID.deleteSelectedFromLibrary()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `PLAYLIST Element`
- `PLAYLIST.deleteSelected`

PLAYLIST-disabledItemColor

The `disabledItemColor` attribute specifies or retrieves the color of a disabled CD track or of online content when offline.

Syntax

```
elementID.disabledItemColor
```
**PLAYLIST.dropDownBackgroundImage**

The `dropDownBackgroundImage` attribute specifies or retrieves the name of the image that displays in the background of the drop-down list.

**Syntax**

```
elementID.dropDownBackgroundImage
```

**Possible Values**

This attribute is a read/write `String` containing the name of an image file. It has no default value.

**Remarks**

This attribute supports PNG, JPG, BMP, and GIF files. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the `hueShift` and `saturation` attributes.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- PLAYLIST Element
The `dropDownImage` attribute specifies or retrieves the name of the image used for the drop-down list button that is displayed at the right edge of the drop-down list.

**Syntax**

`elementID.dropDownImage`

**Possible Values**

This attribute is a read/write `String` containing the name of an image file. It has no default value.

**Remarks**

This attribute supports PNG, JPG, BMP, and GIF files. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the `hueShift` and `saturation` attributes.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `PLAYLIST Element`
- `PLAYLIST.dropDownBackgroundImage`
- `PLAYLIST.hueShift`
- `PLAYLIST.saturation`
The `dropDownList` attribute specifies or retrieves a value indicating which elements show up in the drop-down list box for a given instance of the Playlist control.

**Syntax**

```
elementID.dropDownList
```

**Possible Values**

This attribute is a read/write `String` containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>showAlbums</td>
<td>Shows albums.</td>
</tr>
<tr>
<td>showAll</td>
<td>Default. Shows all available elements including CD playlists, user playlists, and radio presets.</td>
</tr>
<tr>
<td>showCD</td>
<td>Shows the CD playlist.</td>
</tr>
<tr>
<td>showClips</td>
<td>Shows all clips.</td>
</tr>
<tr>
<td>showCurrent</td>
<td>Shows the current, unsaved playlist.</td>
</tr>
<tr>
<td>showLibrary</td>
<td>Shows only library playlists.</td>
</tr>
<tr>
<td>showRadio</td>
<td>Shows radio presets.</td>
</tr>
<tr>
<td>showQueries</td>
<td>Shows queries.</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `PLAYLIST Element`
PLAYLIST.dropDownToolTip

The dropDownToolTip attribute specifies or retrieves the ToolTip shown when the user hovers over the Playlist control drop-down list.

Syntax

```
.elementID.dropDownToolTip
```

Possible Values

This attribute is a read/write String with a default value of "Display playlists, audio, video, or radio stations".

Remarks

When this attribute is set to "" (empty string), no ToolTip is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element

PLAYLIST.dropDownVisible

The dropDownVisible attribute specifies or retrieves a value indicating whether the Playlist drop-down list is visible.

Syntax

```
.elementID.dropDownVisible
```

© 2000-2003 Microsoft Corporation. All rights reserved.
Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The drop-down is visible.</td>
</tr>
<tr>
<td>false</td>
<td>The drop-down is not visible.</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element

PLAYLIST.editButtonVisible

The editButtonVisible attribute specifies or retrieves a value indicating whether the Playlist edit button is visible.

Syntax

elementID.editButtonVisible

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The playlist edit button is visible.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The playlist edit button is not visible.</td>
</tr>
</tbody>
</table>

Remarks
When you set this attribute to true, the Playlist edit button will appear in the lower-left corner of the Playlist control. Clicking this button displays a menu of options allowing the user to edit, clear, sort, open, save, or copy a playlist.

Requirements

Windows Media Player 9 Series or later.

See Also

- **PLAYLIST Element**

### PLAYLIST.foregroundColor

The **foregroundColor** attribute specifies or retrieves the foreground color.

**Syntax**

```
elementID.foregroundColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value equal to the Windows system text color.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **Color Reference**
- **PLAYLIST Element**

© 2000-2003 Microsoft Corporation. All rights reserved.
PLAYLIST.getNextCheckedItem

The `getNextCheckedItem` method retrieves the index of the next checked item in the playlist following the specified index.

Syntax

```
elementID.getNextCheckedItem(item)
```

Parameters

`item`

Number (long) indicating the index of the item to search after.

Return Values

This method returns a Number (long).

Remarks

When there are no further checked items, this method returns –1.

This method has been replaced by `getNextCheckedItem2`, which supports nested playlists.

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element
- PLAYLIST.getNextCheckedItem2
PLAYLIST.getNextCheckedItem2

The `getNextCheckedItem2` method retrieves the index of the next checked item in the playlist following the specified index.

Syntax

```
elementID.getNextCheckedItem2(item)
```

Parameters

`item`  
Number (long) indicating the index of the item to search after.

Return Values

This method returns a Number (long).

Remarks

This method replaces the `getNextCheckedItem` method in order to work with nested playlists. Use –1 to find the first checked item. When there are no further checked items, this method returns –1.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
- PLAYLIST.getNextCheckedItem

© 2000-2003 Microsoft Corporation. All rights reserved.

---

PLAYLIST.getNextSelectedItem

The `getNextSelectedItem` method retrieves the index of the next selected item in the playlist following the
specified index.

Syntax

```
elementID.getNextSelectedItem(item)
```

Parameters

-item

Number (long) indicating the index of the item to search after.

Return Values

This method returns a Number (long).

Remarks

When there are no further selected items, this method returns −1.

This method has been replaced by `getNextSelectedItem2`, which supports nested playlists.

Requirements

Windows Media Player version 7.0 or later.

See Also

- \[ PLAYLIST Element \]
- \[ PLAYLIST.getNextSelectedItem2 \]

© 2000-2003 Microsoft Corporation. All rights reserved.

---

The `getNextSelectedItem2` method retrieves the index of the next selected item in the playlist following the specified index.

Syntax

```
elementID.getNextSelectedItem2(item)
```
Parameters

*item*

**Number (long)** indicating the index of the item to search after.

Return Values

This method returns a **Number (long)**.

Remarks

This method replaces the **getNextSelectedItem** method in order to work with nested playlists. Use –1 to find the first selected item. When there are no further selected items, -1 is returned.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
- PLAYLIST.getNextSelectedItem

PLAYLIST.hueShift

The **hueShift** attribute specifies or retrieves the amount by which the hue of the drop-down images is shifted.

Syntax

`elementID.hueShift`

Possible Values

This attribute is a read/write **Number (float)** with a value ranging from 0.0 to 360.0 with a default value of 0.0.

Remarks

This attribute changes the hue value of the images specified by the **dropDownBackgroundImage** and
dropDownImage attributes if they have been specified and they refer to 8-bit BMP images.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
- PLAYLIST.dropDownBackgroundImage
- PLAYLIST.dropDownImage
- PLAYLIST.saturation

PLAYLIST.itemCount

The itemCount attribute retrieves the number of items currently displayed in the Playlist control.

Syntax

elementID.itemCount

Possible Values

This attribute is a read-only Number (long).

Remarks

The itemCount property will count the total number of expanded items. For example, if there are two playlists that contain three media clips each, itemCount will return 2 if the playlists are not expanded. If only the first playlist is expanded, itemCount will return 4. If both playlists are expanded, itemCount will return 6.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
PLAYLIST.itemErrorColor

The **itemErrorColor** attribute specifies or retrieves the highlight color that indicates a playlist item that has an error condition.

**Syntax**

```
elementID.itemErrorColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value of "red".

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [Color Reference](#)
- [PLAYLIST Element](#)

PLAYLIST.itemMedia

The **itemMedia** attribute retrieves the **Media** corresponding to the given index in the playlist control.
Syntax

elementID.itemMedia(index)

Parameters

index

Number (long) containing the index of a playlist item.

Possible Values

This attribute is a read-only Media object.

Remarks

The itemMedia property will return media objects that are expanded in the playlist control. For example, if there is a playlist that contains three media clips that is not expanded in the playlist control, itemMedia(0) will return the playlist as the media object. If the playlist is expanded, itemMedia(0) will return the first media clip in the playlist.

Requirements

Windows Media Player 9 Series or later.

See Also

- Media Object
- PLAYLIST Element

PLAYLIST.itemPlayingBackgroundColor

The itemPlayingBackgroundColor attribute specifies or retrieves the background color of the currently playing playlist item.

Syntax

elementID.itemPlayingBackgroundColor

Possible Values
This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value of "#222222".

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- [PLAYLIST Element](#)

---

**PLAYLIST.itemPlayingColor**

The **itemPlayingColor** attribute specifies or retrieves the highlight color that indicates the currently playing item in the playlist.

**Syntax**

```xml
elementID.itemPlayingColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has a default value of "#00FF00".

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- [PLAYLIST Element](#)
PLAYLIST.itemPlaylist

The `itemPlaylist` attribute retrieves the playlist for the media item that is displayed at the given index in the playlist control.

**Syntax**

```
elementID.itemPlaylist(index)
```

**Parameters**

`index`

Number (`long`) containing the index of a playlist item.

**Possible Values**

This attribute is a read-only `Playlist` object.

**Remarks**

The `itemPlaylist` property will return the playlist object that is expanded in the playlist control. For example, if there are two nested playlists that are not expanded in the playlist control, and that contain three media clips each, `itemPlaylist(1)` will return the parent playlist that contains the two nested playlists. If the second playlist is expanded, `itemPlaylist(1)` will the second playlist. If both playlists are expanded, `itemPlaylist(1)` will return the first playlist.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [PLAYLIST Element](#)
- [Playlist Object](#)
The `leftStatus` attribute specifies or retrieves the status text that is displayed on the left side and bottom of the Playlist control.

**Syntax**

`elementID.leftStatus`

**Possible Values**

This attribute is a read/write `String`.

**Remarks**

This attribute can combine any text with specific keywords that will display the desired information, such as the total duration of the playlist. The keywords are surrounded by percentage symbols (%) to keep them distinct from the ordinary text.

The following keywords can be used.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count</code></td>
<td>Number of items in the playlist.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Total size of the playlist.</td>
</tr>
<tr>
<td><code>duration</code></td>
<td>Total duration of the playlist.</td>
</tr>
<tr>
<td><code>XXX</code></td>
<td>Does a <code>getItemInfo</code> on the playlist with <code>XXX</code> being the item to receive.</td>
</tr>
<tr>
<td><code>SelectedSize</code></td>
<td>Total size of the selected entries in the playlist.</td>
</tr>
<tr>
<td><code>SelectedCount</code></td>
<td>Total number of selected entries in the playlist.</td>
</tr>
<tr>
<td><code>SelectedDuration</code></td>
<td>Total duration of the selected entries in the playlist.</td>
</tr>
<tr>
<td><code>CheckedCount</code></td>
<td>Total number of checked tracks in the playlist.</td>
</tr>
<tr>
<td><code>CheckedDuration</code></td>
<td>Total duration of the checked tracks in the playlist.</td>
</tr>
<tr>
<td><code>CheckedSize</code></td>
<td>Total size of the checked tracks in the playlist.</td>
</tr>
<tr>
<td><code>DurationString</code></td>
<td>Displays text that describes the duration as &quot;Total Time&quot; or &quot;Estimated Time&quot;, depending on whether the total values are known. This text is followed by &quot;%duration%&quot;.</td>
</tr>
</tbody>
</table>
CheckedDurationString Displays text that describes the duration for all checked items in the playlist as "Total Time" or "Estimated Time", depending on whether the total values are known. This text is followed by "% duration%".

Example

The value "Total Time: %duration%" for a playlist that contains a total duration of seven minutes will display Total Time: 07:00.

Requirements

Windows Media Player 9 Series or later.

See Also

- **PLAYLIST Element**

PLAYLIST.moveSelectedDown

The `moveSelectedDown` method moves the selected item down one position in the list.

Syntax

```
elementID.moveSelectedDown()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.
PLAYLIST.moveSelectedUp

The `moveSelectedUp` method moves the selected item up one position in the list.

Syntax

```
elementID.moveSelectedUp()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- [PLAYLIST Element](#)
- [PLAYLIST.moveSelectedDown](#)
**PLAYLIST.playlist**

The `playlist` attribute specifies or retrieves the `Playlist` object that the Playlist control provides an interface to.

**Syntax**

```
elementID.playlist
```

**Possible Values**

This attribute is a read/write `Playlist` object with no default value.

**Remarks**

If the playlist specified is invalid or if no value is specified, the Playlist control displays the currently playing media item.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [PLAYLIST Element](#)
- [Playlist Object](#)

---

**PLAYLIST.playlistItemsVisible**

The `playlistItemsVisible` attribute specifies or retrieves a value indicating whether the Playlist items area is visible.

**Syntax**

```
```
elementID.playlistItemsVisible

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Playlist items area is visible.</td>
</tr>
<tr>
<td>false</td>
<td>Playlist items area is not visible.</td>
</tr>
</tbody>
</table>

Remarks

The Playlist items area includes the column headers, the contents of the columns, and the scrollbars (if present).

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element

© 2000-2003 Microsoft Corporation. All rights reserved.

---

PLAYLIST.rightStatus

The rightStatus attribute specifies or retrieves the status text that is displayed on the right side and bottom of the Playlist control.

Syntax

```
elementID.rightStatus
```

Possible Values

This attribute is a read/write String.

Remarks
This attribute can combine any text with specific keywords that will display the desired information, such as the total duration of the playlist. The keywords are surrounded by percentage symbols (%) to keep them distinct from the ordinary text.

The following keywords can be used.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of items in the playlist.</td>
</tr>
<tr>
<td>size</td>
<td>Total size of the playlist.</td>
</tr>
<tr>
<td>duration</td>
<td>Total duration of the playlist.</td>
</tr>
<tr>
<td>XXX</td>
<td>Does a <strong>getItemInfo</strong> on the playlist with <code>XXX</code> being the item to receive.</td>
</tr>
<tr>
<td>SelectedSize</td>
<td>Total size of the selected entries in the playlist.</td>
</tr>
<tr>
<td>SelectedCount</td>
<td>Total number of selected entries in the playlist.</td>
</tr>
<tr>
<td>SelectedDuration</td>
<td>Total duration of the selected entries in the playlist.</td>
</tr>
<tr>
<td>CheckedCount</td>
<td>Total number of checked tracks in the playlist.</td>
</tr>
<tr>
<td>CheckedDuration</td>
<td>Total duration of the checked tracks in the playlist.</td>
</tr>
<tr>
<td>CheckedSize</td>
<td>Total size of the checked tracks in the playlist.</td>
</tr>
<tr>
<td>DurationString</td>
<td>Displays text that describes the duration as &quot;Total Time&quot; or &quot;Estimated Time&quot;, depending on whether the total values are known. This text is followed by &quot;%duration%&quot;.</td>
</tr>
<tr>
<td>CheckedDurationString</td>
<td>Displays text that describes the duration for all checked items in the playlist as &quot;Total Time&quot; or &quot;Estimated Time&quot;, depending on whether the total values are known. This text is followed by &quot;%duration%&quot;.</td>
</tr>
</tbody>
</table>

**Example**

The value "Total Time: %duration%" for a playlist that contains a total duration of seven minutes will display **Total Time: 07:00**.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- [PLAYLIST Element](#)
PLAYLIST.saturation

The saturation attribute specifies or retrieves the saturation value of the drop-down images.

Syntax

elementID.saturation

Possible Values

This attribute is a read/write Number (float) with a value ranging from 0.0 to 2.0 with a default value of 1.0.

Remarks

This attribute changes the saturation value of the images specified by the dropDownBackgroundImage and dropDownImage attributes if they have been specified and they refer to 8-bit BMP images.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
- PLAYLIST.dropDownBackgroundImage
- PLAYLIST.dropDownImage
- PLAYLIST.hueShift
The `setCheckedState` method specifies that the indexed item in the playlist is checked.

**Syntax**

```
elementID.setCheckedState(item)
```

**Parameters**

`item`  
Number (`long`) indicating the index of the playlist item to be checked.

**Return Values**

This method returns a `Boolean`.

**Remarks**

You can set all items to the checked state by specifying –1 in the `item` parameter.

This method has been replaced by `setCheckedState2`, which supports nested playlists.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- PLAYLIST Element
- PLAYLIST.setCheckedState2

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**PLAYLIST.setCheckedState2**

The `setCheckedState2` method sets the checked state of the item with the specified index in the Playlist control.

**Syntax**

```
elementID.setCheckedState2(item, checked)
```
Parameters

item

Number (long) indicating the index of the playlist item to be checked or unchecked.

checked

Boolean indicating whether the specified item is to be checked (true) or unchecked (false).

Return Values

This method returns a Boolean.

Remarks

This method replaces the setCheckedState method in order to work with nested playlists. You can set all items to the given state by specifying –1 in the item parameter.

Requirements

Windows Media Player 9 Series or later.

See Also

- PLAYLIST Element
- PLAYLIST.setCheckedState

© 2000-2003 Microsoft Corporation. All rights reserved.

PLAYLIST.setColumnResizeMode

The setColumnResizeMode method specifies how the indexed column sizes itself.

Syntax

\[ elementID.setColumnResizeMode(\text{column, mode}) \]

Parameters

\text{column}
Number (long) indicating the index of the column to be changed.

mode

String indicating the sizing mode. Contains one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutosizeHeader</td>
<td>The column resizes to accommodate all data in both the column and the header.</td>
</tr>
<tr>
<td>AutosizeData</td>
<td>The column resizes to accommodate all data in the column only.</td>
</tr>
<tr>
<td>Fixed</td>
<td>The column is a fixed size.</td>
</tr>
<tr>
<td>Stretches</td>
<td>The column resizes to use the remaining space in the Playlist control after all other columns are resized.</td>
</tr>
</tbody>
</table>

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element

PLAYLIST.setColumnWidth

The setColumnWidth method specifies the column width and changes the resize mode of the column to "Fixed".

Syntax

```
elementID.setColumnWidth(column, width)
```

Parameters
column

**Number (long)** indicating the index of the column being changed.

width

**Number (long)** indicating the new width in pixels.

**Return Values**

This method does not return a value.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- PLAYLIST Element
- PLAYLIST.setColumnResizeMode

---

**PLAYLIST setSelectedState**

The **setSelectedState** method specifies that the indexed item in the playlist is selected.

**Syntax**

```
elementID.setSelectedState(item)
```

**Parameters**

**item**

**Number (long)** indicating the index of an item in the playlist.

**Return Values**

This method does not return a value.
**Remarks**

You can set all items to the selected state by specifying –1 in the *item* parameter.

This method has been replaced by `setSelectedState2`, which supports nested playlists.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [PLAYLIST Element](#)
- [PLAYLIST.setSelectedState2](#)

---

**PLAYLIST.setSelectedState2**

The `setSelectedState2` method sets the selected state of the item with the specified index in the Playlist control.

**Syntax**

```
elementType.setSelectedState2(item, selected)
```

**Parameters**

- **item**
  
  Number (long) indicating the index of an item in the playlist.

- **selected**
  
  Boolean indicating whether the specified item is to be selected (true) or unselected (false).

**Return Values**

This method does not return a value.

**Remarks**
This method replaces the `setSelectedState` method in order to work with nested playlists. You can set all items to the given state by specifying –1 in the item parameter.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- `PLAYLIST` Element
- `PLAYLIST.setSelectedState`

---

**PLAYLIST.sortColumn**

The `sortColumn` method sorts the data in the specified column.

**Syntax**

```
elementID.sortColumn(column)
```

**Parameters**

- `column`

  `Number (long)` indicating the index of the column to sort.

**Return Values**

This method does not return a value.

**Remarks**

This method sorts the specified column in the same way as the column header buttons in the playlist control. If the column has not yet been sorted, it is sorted in alphanumeric order. If it has been sorted, its order is reversed.

For this method to work, the `allowColumnSorting` attribute must be set to true.

**Requirements**

© 2000-2003 Microsoft Corporation. All rights reserved.
Windows Media Player version 7.0 or later.

See Also

- PLAYLIST Element
- PLAYLIST.allowColumnSorting

### PLAYLIST.statusColor

The `statusColor` attribute specifies or retrieves the color of the status line in the Playlist control.

**Syntax**

```xml
elementID.statusColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value. It has a default value equal to the value of the `backgroundColor` attribute.

**Requirements**

Windows Media Player 9 Series or later.

See Also

- Color Reference
- PLAYLIST Element
DROPDOWNPLAYLIST

This is a predefined PLAYLIST element with the following default values.

playlistItemsVisible="false"

Remarks

This will create a simple dropdown PLAYLIST with no items displayed. All properties of this PLAYLIST can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- PLAYLIST Element

ITEMSPLAYLIST

This is a predefined PLAYLIST element with the following default values.

backgroundColor="black"
columns="name=Name;Duration=Time"
columnsVisible="false"
dropDownVisible="false"
foregroundColor="white"

Remarks

This will create a simple PLAYLIST that displays the items in a playlist with no drop-down or column headers visible. All properties of this PLAYLIST can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.
POPUP Element

The POPUP element provides a way for users to select items from a list. These items can be specified by using ITEM elements as children of the POPUP element. The POPUP element is identical to the LISTBOX element except the default value of the popUp attribute is true. When popUp is set to true, the control is displayed only when the show method is called, and its width is set to the width of the content (the ambient width attribute is ignored). The control is hidden automatically when the user selects an item from the list or when the dismiss method is called.

Note  This element requires Windows Media Player for Windows XP or later.

See Also

- LISTBOX Element
- LISTBOX.dismiss
- LISTBOX.popUp
- LISTBOX.show
- Skin Programming Reference

PROGRESSBAR Element

The PROGRESSBAR element provides a way to display progress information in a horizontal or vertical control. This element is identical to the SLIDER element. A progress bar is normally a slider that is not
interactive, but there is nothing that prevents it from being used interactively. This element is provided solely as a convenience tag for making the noninteractivity clear in your code, if you choose to use it that way. It has no other value.

See Also

- Skin Programming Reference
- SLIDER Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoStart</td>
<td>Specifies or retrieves a value indicating whether the current media item begins playing automatically.</td>
</tr>
<tr>
<td>balance</td>
<td>Specifies or retrieves the current stereo balance.</td>
</tr>
<tr>
<td>baseURL</td>
<td>Specifies or retrieves the base URL used for relative path resolution.</td>
</tr>
<tr>
<td>defaultFrame</td>
<td>Specifies or retrieves the name of the frame used to display a URL received in a ScriptCommand.</td>
</tr>
<tr>
<td>enableErrorDialogs</td>
<td>Specifies or retrieves a value indicating whether error dialogs are shown automatically.</td>
</tr>
<tr>
<td>invokeURLs</td>
<td>Specifies or retrieves a value indicating whether URL events should start a Web browser.</td>
</tr>
<tr>
<td>mute</td>
<td>Specifies or retrieves a value indicating whether audio is muted.</td>
</tr>
<tr>
<td>playCount</td>
<td>Specifies or retrieves the number of times a media item will play.</td>
</tr>
<tr>
<td>rate</td>
<td>Specifies or retrieves the current playback rate</td>
</tr>
<tr>
<td>volume</td>
<td>Specifies or retrieves the current volume.</td>
</tr>
</tbody>
</table>

See Also

- Settings Object
The **SLIDER** element provides a way to create and manipulate a simple horizontal or vertical slider control. It supports the following attributes and event handlers. Predefined **SLIDER** elements are also provided for convenience.

The **SLIDER** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>backgroundColor</strong></td>
<td>Specifies or retrieves the background color of the slider control.</td>
</tr>
<tr>
<td><strong>backgroundEndColor</strong></td>
<td>Specifies or retrieves the background ending color of the slider control.</td>
</tr>
<tr>
<td><strong>backgroundHoverImage</strong></td>
<td>Specifies or retrieves the background image of the slider that appears when hovering over it with the mouse.</td>
</tr>
<tr>
<td><strong>backgroundImage</strong></td>
<td>Specifies or retrieves the default background image of the slider.</td>
</tr>
<tr>
<td><strong>borderSize</strong></td>
<td>Specifies or retrieves the border size in pixels.</td>
</tr>
<tr>
<td><strong>cursor</strong></td>
<td>Specifies or retrieves a value indicating which type of cursor appears when the mouse is over the slider control.</td>
</tr>
<tr>
<td><strong>direction</strong></td>
<td>Specifies or retrieves the direction that slider images are laid out.</td>
</tr>
<tr>
<td><strong>disabledColor</strong></td>
<td>Specifies or retrieves the disabled color of the slider control.</td>
</tr>
<tr>
<td><strong>disabledImage</strong></td>
<td>Specifies or retrieves the image of the slider that appears when the control is disabled.</td>
</tr>
<tr>
<td><strong>foregroundColor</strong></td>
<td>Specifies or retrieves the foreground color of the slider control.</td>
</tr>
<tr>
<td><strong>foregroundEndColor</strong></td>
<td>Specifies or retrieves the foreground ending color of the slider control.</td>
</tr>
<tr>
<td><strong>foregroundHoverImage</strong></td>
<td>Specifies or retrieves the foreground image of the slider that appears when hovering over it with the mouse.</td>
</tr>
<tr>
<td><strong>foregroundImage</strong></td>
<td>Specifies or retrieves the default foreground image of the slider.</td>
</tr>
<tr>
<td><strong>foregroundProgress</strong></td>
<td>Specifies or retrieves the current position of the foreground progress bar as a percentage of the slider area.</td>
</tr>
</tbody>
</table>
The SLIDER element can implement the following event handlers.

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onDragBegin</td>
<td>Handles an event that occurs when the user clicks and holds the left mouse button down and begins to drag the mouse.</td>
</tr>
<tr>
<td>onDragEnd</td>
<td>Handles an event that occurs when the left mouse button is released after a dragging operation.</td>
</tr>
<tr>
<td>onPositionChange</td>
<td>Handles an event that occurs when the position of the slider changes as a result of the user clicking or dragging.</td>
</tr>
</tbody>
</table>

The SLIDER element supports the ambient attributes and can implement the ambient event handlers. For more information, see Ambient Attributes and Ambient Event Handlers.

Predefined sliders are normal SLIDER elements with various common attribute settings specified by default. The following predefined sliders are available.

<table>
<thead>
<tr>
<th>Predefined SLIDER</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCESLIDER</td>
<td>A SLIDER used to set audio balance.</td>
</tr>
<tr>
<td>SEEKSLIDER</td>
<td>A SLIDER used to seek to any position within a media file.</td>
</tr>
</tbody>
</table>
The `backgroundColor` attribute specifies or retrieves the background color of the slider control.

### Syntax

```
elementID.backgroundColor
```

### Possible Values

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value. It has no default value.

### Remarks

A basic slider control can be constructed by specifying `backgroundColor` or `backgroundImage`, and `foregroundColor` or `foregroundImage`.

When using the colors, the dimensions of the slider control define the area filled by the background color. The foreground color covers the background color as the slider position increases.

To make a gradient fill of the area occupied by the background or foreground color, specify the `backgroundColorEndColor` or `foregroundColorEndColor` attributes.

### Example

See the `CUSTOMSLIDER.positionImage` attribute for a sample illustrating how the attributes of the `SLIDER` element are used.

### Requirements

Windows Media Player version 7.0 or later.
See Also

- Color Reference
- SLIDER Element
- SLIDER.backgroundEndColor
- SLIDER.backgroundImage
- SLIDER.foregroundColor
- SLIDER.foregroundEndColor
- SLIDER.foregroundImage

© 2000-2003 Microsoft Corporation. All rights reserved.

---

SLIDER.backgroundEndColor

The **backgroundEndColor** attribute specifies or retrieves the background ending color of the slider control.

**Syntax**

`elementID.backgroundEndColor`

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has no default value.

**Remarks**

The **backgroundEndColor** is used in conjunction with a **backgroundColor**. The effect created is a gradient fade from the **backgroundColor** to the **backgroundEndColor**.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Color Reference
- SLIDER Element
- SLIDER.backgroundColor
The `backgroundHoverImage` attribute specifies or retrieves the background image of the slider that appears when hovering over it with the mouse.

**Syntax**

```plaintext
elementID.backgroundHoverImage
```

**Possible Values**

This attribute is a read/write `String` containing the name of an image file.

**Remarks**

This attribute is optional. If it is not specified, the `backgroundImage` will be used.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [SLIDER Element](#)
The `backgroundImage` attribute specifies or retrieves the background image of the slider.

**Syntax**

```
elementID.backgroundImage
```

**Possible Values**

This attribute is a read/write `String` containing the name of an image file.

**Remarks**

This attribute is optional. When using images to construct a slider, the `backgroundImage` is used for the main slider image. The `thumbImage` represents the actual slider and can be moved using the mouse. At the `thumbImage` slider there is an invisible line where the background image is displayed on one side of the line, and the foreground image is displayed on the other side.

When the `thumbImage` slider is moved with the mouse, if `slide` is set to true, the foreground image slides along as if being pulled by the slider to cover the background image. If `slide` is set to false, the foreground image does not move, but is revealed in place, as if the slider is moving the background image off the foreground image.

If the `tiled` attribute is set to true and the background image is smaller than the slider control, the image will be tiled either horizontally or vertically depending on the `direction` attribute. If the `borderSize` attribute is set to a value greater than zero, the number specified will be the number of pixels from the left and right or top and bottom of the image (again, depending on the `direction` attribute) that will be reserved for the borders of the slider control. In this case, only the central portion of the image is used for tiling the remainder of the control.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `SLIDER` Element
- `SLIDER.foregroundImage`
- `SLIDER.slide`
- `SLIDER.thumbImage`
SLIDER.borderSize

The `borderSize` attribute specifies or retrieves the border width in pixels.

Syntax

```
elementID.borderSize
```

Possible Values

This attribute is a read/write `Number (long)` representing the border width in pixels. The default value is zero.

Remarks

This attribute defines an offset from the beginning and end of the slider control—that is, from the left and right if the `direction` attribute is set to "horizontal", and from the top and bottom if it is set to "vertical". These offset positions dictate the minimum and maximum positions of the slider thumb, beyond which the foreground color or image will not be applied.

If a background image is used with the `tiled` attribute set to true, the offset is applied to the image, dictating the amount of the image (either from the left and right or from the top and bottom) to be used for the beginning and end borders of the slider control, with the central portion of the image being tiled throughout the remainder. In this way, a small background image can be used to cover the full length of a larger slider control.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `SLIDER Element`
- `SLIDER.foregroundColor`
- `SLIDER.foregroundImage`
- `SLIDER.thumbImage`
- `SLIDER.tiled`

SLIDER.cursor

The `cursor` attribute specifies or retrieves a value indicating which cursor appears when the mouse is over the
Slider control.

**Syntax**

elementID.cursorelementID.cursorelementID.cursorelementID.cursorelementID.cursorelementID.cursorelementID.cursor

**Possible Values**

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Default. Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
</tbody>
</table>

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [SLIDER Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**SLIDER.direction**
The **direction** attribute specifies or retrieves the direction that slider images are laid out.

**Syntax**

```html
elementID.direction
```

**Possible Values**

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal</td>
<td>Default. The minimum position is at the left, and the maximum position is at the right.</td>
</tr>
<tr>
<td>vertical</td>
<td>The minimum position is at the bottom, and the maximum position is at the top.</td>
</tr>
</tbody>
</table>

**Remarks**

The fill or movement of the **foregroundColor** or **foregroundImage** is along the vertical or horizontal axis according to the specified value.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `SLIDER.Element`
- `SLIDER.foregroundColor`
- `SLIDER.foregroundImage`

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**SLIDER.disabledColor**

The **disabledColor** attribute specifies or retrieves the color of the slider control when it is disabled.

**Syntax**

```html
elementID.disabledColor
```
Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value. It has no default value.

Remarks

When the slider control is specified using foreground and background colors, the disabled color specifies the color of the control when enabled is set to false.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Color Reference
- SLIDER Element
- AmbientAttributes.enabled

SLIDER.disabledImage

The disabledImage attribute specifies or retrieves the image of the slider that is used when the slider control is disabled.

Syntax

`elementID.disabledImage`

Possible Values

This attribute is a read/write String containing the name of an image file.

Remarks

The disabledImage is optional. If it is not provided, the backgroundImage is used for all disabled states. When a slider control is disabled, no foreground image is visible.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).
Requirements

Windows Media Player version 7.0 or later.

See Also

- SLIDER Element
- SLIDER.backgroundImage

SLIDER.foregroundColor

The foregroundColor attribute specifies or retrieves the foreground color of the slider control.

Syntax

elementID.foregroundColor

Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value. The default value is "white".

Remarks

A basic slider control can be constructed by specifying one of two pairs of attributes: the backgroundColor and foregroundColor, or the backgroundImage and foregroundImage.

When you construct a slider control using the color attributes, the dimensions of the slider control define the area filled by the background color. The foreground color covers the background color as the slider position increases.

To make a gradient fill in the area occupied by the background or foreground color, specify the backgroundColorEndColor or foregroundEndColor attributes.

Example

See the CUSTOMSLIDER.positionImage attribute for a sample illustrating how the attributes of the SLIDER element are used.
The `foregroundEndColor` attribute specifies or retrieves the foreground ending color of the slider control.

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value. It has no default value.

**Remarks**

The `foregroundEndColor` is used in conjunction with a `foregroundColor`. The effect created is a gradient fade from the `foregroundColor` to the `foregroundEndColor`.

**Requirements**

Windows Media Player version 7.0 or later.
The `foregroundHoverImage` attribute specifies or retrieves the foreground image of the slider that appears when hovering over it with the mouse.

**Syntax**

```
elementID.foregroundHoverImage
```

**Possible Values**

This attribute is a read/write `String` containing the name of an image file.

**Remarks**

The `foregroundHoverImage` is optional. If it is not provided, the `backgroundImage` is used.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [SLIDER Element](#)
The **foregroundImage** attribute specifies or retrieves the foreground image of the slider.

### Syntax

```
elementID.foregroundImage
```

### Possible Values

This attribute is a read/write **String** containing the name of an image file.

### Remarks

This attribute is optional. When using images to construct a slider control, the **backgroundImage** is used for the main slider image. The **thumbImage** represents the actual slider and can be moved using the mouse. At the **thumbImage** slider there is an invisible line where the background image is displayed on one side of the line, and the foreground image is displayed on the other side.

When the **thumbImage** slider is moved with the mouse, if **slide** is set to true, the foreground image slides along as if being pulled by the slider to cover the background image. If **slide** is set to false, the foreground image does not move, but is revealed in place, as if the slider is moving the background image off the foreground image.

If the **tiled** attribute is set to true and the foreground image is smaller than the foreground area of the slider control, the image will be tiled either horizontally or vertically, depending on the **direction** attribute, to fill the available space.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

### Requirements

Windows Media Player version 7.0 or later.

See Also

- **SLIDER Element**
- **SLIDER.backgroundImage**
- **SLIDER.slide**
- **SLIDER.thumbImage**
- **SLIDER.value**
SLIDER.foregroundProgress

The `foregroundProgress` attribute specifies or retrieves the current position of the foreground progress bar as a percentage of the slider area.

**Syntax**

```
elementID.foregroundProgress
```

**Possible Values**

This attribute is a read/write `Number (float)` ranging from 0 to 100.

**Example**

```
<SLIDER
   id="seek"
   backgroundColor="blue"
   foregroundColor="red"
   thumbImage="seekthumb.bmp"
   min="0"
   max="wmpprop:player.currentMedia.duration"
   value="wmpprop:player.controls.currentPosition"
   useForegroundProgress="true"
   foregroundProgress="wmpprop:player.network.downloadProgress"
   ondragend="player.controls.currentposition=value"
/>
```

**Remarks**

This attribute is used primarily to track the download progress of a media file while simultaneously tracking the current play position of the file using the `value` attribute. The position of the slider thumb is constrained to the area of the foreground progress. This allows interactive seeking to take place only within the available portion of a downloading file.

To use this functionality, the `useForegroundProgress` attribute must be set to true.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- SLIDER Element
- SLIDER.min
- SLIDER.max
- SLIDER.useForegroundProgress
- SLIDER.value
SLIDER.max

The **max** attribute specifies or retrieves the maximum value of the range defined by the slider control.

**Syntax**

```
elementID.max
```

**Possible Values**

This attribute is a read/write **Number (float)** with a default value of 100.

**Remarks**

The value specified for **max** must be greater than the one for **min**.

**Example**

See the `CUSTOMSLIDER.positionImage` attribute for a sample illustrating how the attributes of the **SLIDER** element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **SLIDER Element**
- **SLIDER.min**
- **SLIDER.value**
SLIDER.min

The min attribute specifies or retrieves the minimum value of the range defined by the slider control.

Syntax

elementID.min

Possible Values

This attribute is a read/write Number (float) with a default value of zero.

Remarks

The value specified for min must be less than the one for max.

Example

See the CUSTOMSLIDER.positionImage attribute for a sample illustrating how the attributes of the SLIDER element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- SLIDER Element
- SLIDER.max
- SLIDER.value

SLIDER.onDragBegin

The onDragBegin event handler handles an event that occurs when the user clicks and holds the left mouse button down and begins to drag the mouse.

Syntax

onDragBegin
Requirements

Windows Media Player version 7.0 or later.

See Also

- **SLIDER Element**

---

**SLIDER.onDragEnd**

The **onDragEnd** event handler handles an event that occurs when the left mouse button is released after a dragging operation.

**Syntax**

```javascript
onDragEnd
```

**Requirements**

Windows Media Player version 7.0 or later.

See Also

- **SLIDER Element**

---

**SLIDER.onPositionChange**
The `onPositionChange` event handler handles an event that occurs when the position of the slider changes as a result of the user clicking or dragging.

**Syntax**

```
onPositionChange
```

**Remarks**

If the position of the slider changes as a result of the `value` attribute being modified in script, this event is not fired. To accommodate this possibility, implement the `value_onchange` event handler instead.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `SLIDER Element`

---

The `slide` attribute specifies or retrieves a value indicating whether the foreground image slides over the background image or is gradually revealed in a static position over the background image.

**Syntax**

```
elementID.slide
```

**Possible Values**

This attribute is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The foreground image slides over the background image.</td>
</tr>
<tr>
<td>false</td>
<td>The foreground image is revealed in place over the background image.</td>
</tr>
</tbody>
</table>
Remarks

When the thumbImage slider is moved with the mouse, if slide is set to true, the foreground image slides along as if being pulled by the slider to cover the background image. If slide is set to false, the foreground image does not move, but is revealed in place, as if the slider is moving the background image off the foreground image.

Requirements

Windows Media Player version 7.0 or later.

See Also

- SLIDER Element
- SLIDER.backgroundImage
- SLIDER.foregroundImage

SLIDER.thumbDisabledImage

The thumbDisabledImage attribute specifies or retrieves the thumb image of the slider control that is used when the control is disabled.

Syntax

elementID.thumbDisabledImage

Possible Values

This attribute is a read/write String containing the name of an image file.

Remarks

The thumbDisabledImage is optional. If it is not specified, thumbImage is used instead.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

Requirements

Windows Media Player version 7.0 or later.
The `thumbDownImage` attribute specifies or retrieves the image representing the down state of the thumb.

**Syntax**

```plaintext
elementID.thumbDownImage
```

**Possible Values**

This attribute is a read/write `String` containing the name of an image file.

**Remarks**

The `thumbDownImage` is optional and is the image used to represent the down state of the `thumbImage`. If a `thumbDownImage` is not provided, the slider will use the `thumbImage`.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `SLIDER Element`
- `AmbientAttributes.enabled`
- `SLIDER.thumbImage`
SLIDER.thumbHoverImage

The **thumbHoverImage** attribute specifies or retrieves the image of the thumb that appears when hovering over it with the mouse.

**Syntax**

```
elementID.thumbHoverImage
```

**Possible Values**

This attribute is a read/write **String** containing the name of an image file.

**Remarks**

The **thumbHoverImage** is optional. If it is not provided, the slider control will not change appearance when the mouse hovers over it.

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [SLIDER Element](#)

---

SLIDER.thumbImage

The **thumbImage** attribute specifies or retrieves the image that will be used to represent the current position of
the slider.

Syntax

elementID.thumbImage

Possible Values

This attribute is a read/write String containing the name of an image file.

Remarks

The thumbImage specifies the image that will be used to represent current position, as well as indicate that the user can take action with the control. If no thumb image is specified, the slider is non-interactive.

The thumb image is centered in the narrow dimension of the slider control. If the thumb image is narrower than the control, the image appears in the middle of the background. If the thumb image is larger than the control, the ends of the image are cut off.

The position of the slider is specified by the center of the thumb image. If borderBottomSize is zero, only half the thumb image will be visible at the beginning and end slider positions. To prevent this, set borderBottomSize to a value greater than or equal to half the width of the thumb image (or half the height if direction is set to "vertical").

The supported formats are BMP, JPG, PNG, and GIF (not including animated GIFs).

Example

See the CUSTOMSLIDER,positionImage attribute for a sample illustrating how the attributes of the SLIDER element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- SLIDER Element
- SLIDER.borderSize
- SLIDER.direction
The **tiled** attribute specifies or retrieves a value indicating whether the slider image will be tiled.

**Syntax**

```
.elementID.tiled
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The image bitmap will be repeated until it fills the entire region of the control.</td>
</tr>
<tr>
<td>false</td>
<td>The image will not be tiled.</td>
</tr>
</tbody>
</table>

**Remarks**

This attribute applies only if you are using foreground and background images to define a slider control. If the images are smaller than the defined area of the slider, and the **tiled** attribute is set to true, the images will be repeated until they fill the entire length of the control.

You may wish to use this attribute in conjunction with the **borderSize** attribute. The **borderSize** attribute allows you to define a border that is not repeated during tiling.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **SLIDER Element**
- **SLIDER.backgroundImage**
- **SLIDER.borderSize**
- **SLIDER.foregroundImage**
SLIDER.toolTip

The **toolTip** attribute specifies or retrieves the ToolTip text for the slider control.

**Syntax**

```
elementID.toolTip
```

**Possible Values**

This attribute is a read/write **String** with a maximum length of 1024 characters. It has no default value.

**Remarks**

When this attribute is set to "" (empty string), no ToolTip is displayed.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [SLIDER Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

SLIDER.transparencyColor

The **transparencyColor** attribute specifies or retrieves the transparent color of the slider control background and foreground images.

**Syntax**

```
elementID.transparencyColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value. It has no default value.
Remarks

Any part of the image containing the *transparencyColor* will allow the background to show through.

Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended when *transparencyColor* is used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- **SLIDER Element**
- **Color Reference**

---

**SLIDER.useForegroundProgress**

The *useForegroundProgress* attribute specifies or retrieves a value indicating whether the foreground progress bar will be used.

Syntax

```
elementID.useForegroundProgress
```

Possible Values

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Use foreground progress.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Do not use foreground progress.</td>
</tr>
</tbody>
</table>

Remarks

This attribute allows the use of the *foregroundProgress* attribute, which is used primarily to track the download progress of a media file while simultaneously tracking the current play position of the file using the
**value** attribute.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- SLIDER Element
- SLIDER.foregroundProgress
- SLIDER.value

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**SLIDER.value**

The **value** attribute specifies or retrieves the current position of the slider.

**Syntax**

```plaintext
elementID.value
```

**Possible Values**

This attribute is a read/write **Number (float)** with a default value of **min**.

**Remarks**

The **value** should always be greater than or equal to **min** and less than or equal to **max**. If you specify a value outside this range, **value** and the position of the slider are not changed.

**Example**

See the **CUSTOMSLIDER.positionImage** attribute for a sample illustrating how the attributes of the **SLIDER** element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**
BALANCESLIDER

This is a predefined **SLIDER** with the following default values.

```
toolTip="Balance"
max="100"
min="-100"
value="wmpprop:player.settings.balance"
value_onchange="jscript:player.settings.balance=value;"
```

**Remarks**

This creates a **SLIDER** control that sets the audio balance. The ToolTips are localized. All properties of this **SLIDER** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player 7.0 or later.

**See Also**

- [SLIDER Element](#)

SEEKSLIDER
This is a predefined **SLIDER** with the following default values.

```html
toolTip="Seek"
foregroundProgress="wmprop:player.network.downloadProgress"
max="wmprop:player.currentMedia.duration"
min="0"
value="wmprop:player.controls.currentPosition"
onDragEnd="jscript:player.controls.currentPosition=value;"
useForegroundProgress="true"
```

**Remarks**

This creates a **SLIDER** control that seeks the media file to any position. The ToolTips are localized. All properties of this **SLIDER** can be overridden by explicitly specifying them.

**Requirements**

Windows Media Player 7.0 or later.

**See Also**

- **SLIDER Element**

---

This is a predefined **SLIDER** with the following default values.

```html
toolTip="Volume"
max="100"
min="0"
value="wmprop:player.settings.volume"
value_onchange="jscript:player.settings.volume=value;
player.settings.mute=false;"
```

**Remarks**

This creates a **SLIDER** control that sets the audio volume. The ToolTips are localized. All properties of this **SLIDER** can be overridden by explicitly specifying them.

**Requirements**
SUBVIEW Element

The SUBVIEW element provides a way to manipulate a portion of a skin, for example, to provide a control panel that can be hidden when it is not being used. SUBVIEW elements are always children of parent VIEW elements, and can contain other skin element except for VIEW, THEME, and other SUBVIEW elements.

The SUBVIEW element supports the following attributes, which are defined under the VIEW element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Specifies or retrieves the background color of the SUBVIEW control. The default value is &quot;none&quot;.</td>
</tr>
<tr>
<td>backgroundImage</td>
<td>Specifies or retrieves the background image of the SUBVIEW control.</td>
</tr>
<tr>
<td>backgroundImageHueShift</td>
<td>Specifies or retrieves the amount by which the hue of the background image is shifted.</td>
</tr>
<tr>
<td>backgroundImageSaturation</td>
<td>Specifies or retrieves the saturation value of the background image.</td>
</tr>
<tr>
<td>backgroundTiled</td>
<td>Specifies or retrieves a value indicating whether the background image of the SUBVIEW control is tiled.</td>
</tr>
<tr>
<td>resizeBackgroundImage</td>
<td>Specifies or retrieves a value indicating whether the background image can be resized.</td>
</tr>
<tr>
<td>transparencyColor</td>
<td>Specifies or retrieves the transparency color of the background image.</td>
</tr>
</tbody>
</table>

The SUBVIEW element supports the ambient attributes, except where noted. For more information, see Ambient Attributes.

The SUBVIEW element can implement the following ambient event handlers: onendmove and onresize.
The **TEXT** element provides a way to create and control the appearance of text within a skin by using the following attributes. Predefined **TEXT** elements are also provided for convenience.

The **TEXT** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Specifies or retrieves the background color for the Text control.</td>
</tr>
<tr>
<td>cursor</td>
<td>Specifies or retrieves a value indicating which cursor appears when the mouse is over the Text control.</td>
</tr>
<tr>
<td>disabledBackgroundColor</td>
<td>Specifies or retrieves the background color used for the Text control when it is disabled.</td>
</tr>
<tr>
<td>disabledFontStyle</td>
<td>Specifies or retrieves the font style used for the Text control when it is disabled.</td>
</tr>
<tr>
<td>disabledForegroundColor</td>
<td>Specifies or retrieves the text color used when the Text control is disabled.</td>
</tr>
<tr>
<td>fontFace</td>
<td>Specifies or retrieves the typeface for the Text control.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Specifies or retrieves the font size for the Text control.</td>
</tr>
<tr>
<td>fontSmoothing</td>
<td>Specifies or retrieves a value indicating whether font smoothing is enabled.</td>
</tr>
<tr>
<td>fontStyle</td>
<td>Specifies or retrieves the font style for the Text control.</td>
</tr>
<tr>
<td>foregroundColor</td>
<td>Specifies or retrieves the text color for the Text control.</td>
</tr>
<tr>
<td>hoverBackgroundColor</td>
<td>Specifies or retrieves the background color used for the Text control when the mouse cursor hovers over it.</td>
</tr>
<tr>
<td>hoverFontStyle</td>
<td>Specifies or retrieves the font style used for the Text control when the mouse cursor hovers over it.</td>
</tr>
</tbody>
</table>
The **TEXT** element supports the ambient attributes and can implement the ambient event handlers. For more information, see [Ambient Attributes](#) and [Ambient Event Handlers](#).

Predefined text elements are normal **TEXT** elements with various common attribute settings specified by default. The following predefined text elements are available.

<table>
<thead>
<tr>
<th>Predefined TEXT</th>
<th>Description</th>
</tr>
</thead>
</table>
| CURRENTPOSITIONTEXT | A **TEXT** element with a built in listener for `player.controls.currentPositionString`.
| DURATIONTEXT | A **TEXT** element with a built in listener for `player.currentMedia.DurationString`.
| STATUSTEXT | A **TEXT** element with a built in listener for `player.status`.
| TRACKNAMETEXT | A **TEXT** element with a built in listener for `player.currentMedia.name`.

See Also

- [Skin Programming Reference](#)
TEXT.backgroundColor

The **backgroundColor** attribute specifies or retrieves the background color for the Text control.

**Syntax**

```
elementType.backgroundColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value or the value "none". It has a default value of "none", which means the background is transparent.

**Remarks**

The background color is set for the height and width of the control. If height and width are not set, the background color is set to the height and width of the text.

When you use **alphaBlend** or **alphaBlendTo** with a TEXT element that does not have the **backgroundColor** specified, a background color of black will be used. If the foreground color is also black (which is the default value for the **foregroundColor** attribute), your text may become unreadable. In order to prevent this, always specify the **backgroundColor** attribute, or set **foregroundColor** to a color other than black.

**Example**

See the **value** attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- AmbientAttributes.alphaBlend
- AmbientAttributes.alphaBlendTo
- Color Reference
- TEXT Element
- TEXT.foregroundColor
TEXT.cursor

The **cursor** attribute specifies or retrieves a value indicating which cursor appears when the mouse is over the Text control.

**Syntax**

```
<elementID>.cursor
```

**Possible Values**

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Default. Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani or *.cur</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
</tbody>
</table>

**Example**

See the **value** attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- TEXT Element
The `disabledBackgroundColor` attribute specifies or retrieves the background color used for the Text control when it is disabled.

**Syntax**

```
elementID.disabledBackgroundColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value.

**Remarks**

If the `disabledBackgroundColor` is not specified, the `backgroundColor` is used.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- [TEXT Element](#)
- [TEXT.backgroundColor](#)
The `disabledFontStyle` attribute specifies or retrieves the font style used for the Text control when it is disabled.

### Syntax

```
elementID.disabledFontStyle
```

### Possible Values

This attribute is a read/write `String` containing one or more of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold font style.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic font style.</td>
</tr>
<tr>
<td>Underline</td>
<td>Underline font style.</td>
</tr>
<tr>
<td>Strikeout</td>
<td>Strikeout font style.</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal font style.</td>
</tr>
</tbody>
</table>

### Remarks

Any combination of the values can be used, separated with spaces. The Normal style has priority over all other values, and any others specified along with Normal will be ignored.

If `disabledFontStyle` is not specified, `fontStyle` is used.

### Example

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- `TEXT Element`
- `TEXT.fontStyle`
The `disabledForegroundColor` attribute specifies or retrieves the text color used for the Text control when it is disabled.

**Syntax**

```html
elementID.disabledForegroundColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value.

**Remarks**

If `disabledForegroundColor` is not specified, `foregroundColor` is used.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `Color Reference`
- `TEXT Element`
- `TEXT.foregroundColor`
The **fontFace** attribute specifies or retrieves the typeface for the Text control.

**Syntax**

```
elementID.fontFace
```

**Possible Values**

This attribute is a read/write **String**.

**Remarks**

This attribute can be the name of any valid font available on Windows. Windows Media Player will not support installing fonts, so choose a font that you know will be on the intended system.

If the **fontFace** specified is not available on the user's system, the Text control defaults to the Windows system font.

**Example**

See the **value** attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- TEXT Element
- TEXT.fontSize
TEXT.fontSize

The **fontSize** attribute specifies or retrieves the font size for the Text control.

**Syntax**

```plaintext
elementType.fontSize
```

**Possible Values**

This attribute is a read/write **Number (long)** specifying the font size in points. It has a default value of 10.

**Example**

See the **value** attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- TEXT Element
- TEXT.fontFace

TEXT.fontSmoothing

The **fontSmoothing** attribute specifies or retrieves a value indicating whether font smoothing is enabled.

**Syntax**

```plaintext
elementType.fontSmoothing
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
true Font smoothing is enabled.
false Default. Font smoothing is disabled.

Remarks

Font smoothing uses the anti-aliasing feature of the operating system. If font smoothing is enabled and the operating system supports anti-aliasing, the control attempts to anti-alias the text. Not all fonts support anti-aliasing.

Operating Systems that support anti-aliasing:

- Windows 98
- Windows 2000
- Windows 95 with Plus!
- Windows NT® 4.0

Warning  Font smoothing over a transparent color may cause the transparent color to blend into the characters. It is not recommended that you use fontSmoothing if the text will display over a transparency.

Example

See the value attribute for a sample illustrating how the attributes of the TEXT element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- TEXT Element

© 2000-2003 Microsoft Corporation. All rights reserved.

TEXT.fontStyle

The fontStyle attribute specifies or retrieves the font style for the Text control.

Syntax

elementID.fontStyle
Possible Values

This attribute is a read/write String containing one or more of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold font style.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic font style.</td>
</tr>
<tr>
<td>Underline</td>
<td>Underline font style.</td>
</tr>
<tr>
<td>Strikeout</td>
<td>Strikeout font style.</td>
</tr>
</tbody>
</table>

Remarks

Any combination of the values can be used, separated with spaces. The Normal style has priority over all other values, and any others specified along with Normal will be ignored.

Example

See the value attribute for a sample illustrating how the attributes of the TEXT element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- TEXT Element

© 2000-2003 Microsoft Corporation. All rights reserved.

TEXT.foregroundColor

The foregroundColor attribute specifies or retrieves the text color for the Text control.

Syntax

```
<elementID>.foregroundColor
```
Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value. The default value is "black".

Example

See the value attribute for a sample illustrating how the attributes of the TEXT element are used.

When you use alphaBlend or alphaBlendTo with a TEXT element that does not have the backgroundColor specified, a background color of black will be used. If the foreground color is also black (which is the default value for the foregroundColor attribute), your text may become unreadable. In order to prevent this, always specify the backgroundColor attribute, or set foregroundColor to a color other than black.

Requirements

Windows Media Player version 7.0 or later.

See Also

- AmbientAttributes.alphaBlend
- AmbientAttributes.alphaBlendTo
- Color Reference
- TEXT Element
- TEXT.backgroundColor

TEXT.hoverBackgroundColor

The hoverBackgroundColor attribute specifies or retrieves the background color used for the Text control when the mouse cursor hovers over it.

Syntax

\[elementID.hoverBackgroundColor\]

Possible Values

This attribute is a read/write String containing any Microsoft Internet Explorer color value.

Remarks
If `hoverBackgroundColor` is not specified, the `backgroundColor` is used.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- `TEXT Element`
- `TEXT.backgroundColor`

---

**TEXT.hoverFontStyle**

The `hoverFontStyle` attribute specifies or retrieves the font style used for the Text control when the mouse cursor hovers over it.

**Syntax**

```
elementID.hoverFontStyle
```

**Possible Values**

This attribute is a read/write `String` containing one or more of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold font style.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic font style.</td>
</tr>
<tr>
<td>Underline</td>
<td>Underline font style.</td>
</tr>
<tr>
<td>Strikeout</td>
<td>Strikeout font style.</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal font style.</td>
</tr>
</tbody>
</table>
Remarks

Any combination of the values can be used, separated with spaces. The Normal style has priority over all other values, and any others specified along with Normal will be ignored.

If `hoverFontStyle` is not specified, `fontStyle` is used.

Example

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `TEXT Element`
- `TEXT.fontStyle`

TEXT.hoverForegroundColor

The `hoverForegroundColor` attribute specifies or retrieves the text color used for the Text control when the mouse cursor hovers over it.

Syntax

`elementID.hoverForegroundColor`

Possible Values

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value.

Remarks

If `hoverForegroundColor` is not specified, `foregroundColor` is used.

Example
See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Color Reference
- `TEXT Element`
- `TEXT.foregroundColor`

TEXT.justification

The `justification` attribute specifies or retrieves the alignment of the text within the Text control.

Syntax

`elementID.justification`

Possible Values

This attribute is a read/write `String`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Default. Aligns the text to the left of the Text control.</td>
</tr>
<tr>
<td>Right</td>
<td>Aligns the text to the right of the Text control.</td>
</tr>
<tr>
<td>Center</td>
<td>Aligns the text to the horizontal center of the Text control.</td>
</tr>
</tbody>
</table>

Example

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

Requirements

Windows Media Player version 7.0 or later.
TEXT.scrolling

The **scrolling** attribute specifies or retrieves a value indicating whether the text scrolls.

**Syntax**

```plaintext
elementID.scrolling
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Scrolling is enabled.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Scrolling is disabled.</td>
</tr>
</tbody>
</table>

**Remarks**

The scrolling feature provides a two-space buffer between the end of the text and the beginning of the repeated line.

The **justification** attribute specifies where the text first appears before the scrolling begins.

**Example**

See the **value** attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- TEXT Element
The `scrollingAmount` attribute specifies or retrieves the number of pixels that the text moves during each scrolling movement.

**Syntax**

```
elementID.scrollingAmount
```

**Possible Values**

This attribute is a positive read/write `Number (int)` with a default value of 6.

**Remarks**

For smooth scrolling, `scrollingAmount` should be small. For fast drawing with big gaps, the `scrollingAmount` should be larger. If `scrolling="false"`, `scrollingAmount` is ignored.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the TEXT element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- TEXT Element
- TEXT.justification
- TEXT.scrolling
- TEXT.scrollingAmount
- TEXT.scrollingDelay
- TEXT.scrollingDirection
The `scrollingDelay` attribute specifies or retrieves the time delay between scrolling movements.

**Syntax**

```xml
elementID.scrollingDelay
```

**Possible Values**

This attribute is a read/write `Number (int)` specifying the delay in milliseconds. It has a minimum value of 30, and a default value of 85. If a value less than the minimum is specified, the default is used.

**Remarks**

If `scrolling` is set to false, `scrollingDelay` is ignored.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `TEXT Element`
- `TEXT.scrolling`
TEXT.scrollingDirection

The `scrollingDirection` attribute specifies or retrieves the direction of the scrolling text.

**Syntax**

```
elementID.scrollingDirection
```

**Possible Values**

This attribute is a read/write `String`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Default. Scroll from right to left.</td>
</tr>
<tr>
<td>Right</td>
<td>Scroll from left to right.</td>
</tr>
</tbody>
</table>

**Remarks**

If `scrolling` is set to false, `scrollingDirection` is ignored.

**Example**

See the `value` attribute for a sample illustrating how the attributes of the `TEXT` element are used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [TEXT Element](#)
- [TEXT.scrolling](#)

TEXT.textWidth

The `textWidth` attribute retrieves the width in pixels of the text contained in the `value` attribute.
Syntax

\texttt{elementID.textWidth}

Possible Values

This attribute is a read-only \texttt{Number (int)}.

Remarks

The value returned is based on the \texttt{fontFace}, \texttt{fontSize}, and \texttt{fontStyle} attributes as well as on the actual characters in the \texttt{value} attribute string.

Example

See the \texttt{value} attribute for a sample illustrating how the attributes of the \texttt{TEXT} element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- \texttt{TEXT Element}
- \texttt{TEXT.fontFace}
- \texttt{TEXT.fontSize}
- \texttt{TEXT.fontStyle}
- \texttt{TEXT.value}

© 2000-2003 Microsoft Corporation. All rights reserved.

---

\texttt{TEXT.toolTip}

The \texttt{toolTip} attribute specifies or retrieves the ToolTip text for the text control.

Syntax

\texttt{elementID.toolTip}

Possible Values

This attribute is a read/write \texttt{String} with a maximum length of 1024 characters. It has no default value.
Remarks

If this attribute is not specified, and the text in the value attribute is truncated in the Text control, or wordWrap is set to true, the ToolTip will display the full text of the value attribute.

When this attribute is set to "" (empty string), no ToolTip is displayed.

Example

See the value attribute for a sample illustrating how the attributes of the TEXT element are used.

Requirements

Windows Media Player version 7.0 or later.

See Also

- TEXT Element
- TEXT.value
- TEXT.wordWrap

TEXT.value

The value attribute specifies or retrieves the text that is displayed in the Text control.

Syntax

elementID.value

Possible Values

This attribute is a read/write String.

Remarks

If the width of the Text control is insufficient to contain the text provided, the text is cropped, and an ellipsis is shown to illustrate the fact. If the toolTip attribute has not been set, the complete text will then appear as a ToolTip when the cursor hovers over the control.
If a width is not specified, the default width for the control is the width of the string.

If the height of the control is not specified, the default height is one line.

If the \texttt{wordWrap} attribute is set to true and the height of the control is enough to accommodate another line of text, the text wraps to a subsequent line. Wrapping only occurs between words. Line breaks may also be forced, as explained under \texttt{wordWrap}.

\textbf{Example}

The following sample is a complete skin definition file that illustrates how the attributes of the \texttt{TEXT} element are used. It can be found in the Samples directory that was installed with the SDK.

\begin{verbatim}
<THEME>
  <VIEW
    height = "175"
  >
    <TEXT
      width = "150"
      fontSize = "30"
      hoverFontStyle = "Bold"
      hoverForegroundColor = "red"
      disabledForegroundColor = "#CCCCCC"
      justification = "Center"
      value = "Play"
      cursor = "hand"
      enabled = "wmpenabled:player.controls.play"
      onClick = "JScript: player.URL='http://proseware.com/laure.wma';"
    />
    <TEXT
      top = "50"
      width = "150"
      fontSize = "30"
      hoverFontStyle = "Bold"
      hoverForegroundColor = "red"
      disabledForegroundColor = "#CCCCCC"
      justification = "Center"
      value = "Stop"
      cursor = "hand"
      enabled = "wmpenabled:player.controls.stop"
      onClick = "JScript: player.controls.stop();"
    />
    <TEXT
      top = "100"
      width = "150"
      fontSize = "30"
      hoverFontStyle = "Bold"
      hoverForegroundColor = "red"
      disabledForegroundColor = "#CCCCCC"
      justification = "Center"
      value = "Close"
      cursor = "hand"
      onClick = "JScript: view.close();"
    />
    <TEXT
      top = "30"
      left = "120"
      width = "200"
      fontSize = "20"
      fontStyle = "Underline"
      justification = "Center"
  </VIEW>
</THEME>
\end{verbatim}
value = "Volume"
 />
<TEXT
top = "60"
left = "120"
width = "200"
fontSize = "40"
j ustification = "Center"
value = "wmpprop:player.settings.volume"
 />
<TEXT
top = "65"
left = "142"
width = "40"
fontSize = "30"
hoverFontStyle = "Bold"
hoverForegroundColor = "red"
j ustification = "Center"
value = "-"
cursor = "hand"
toolTip = "decrease volume"
onClick = "player.settings.volume = player.settings.volume - 5"
 />
<TEXT
top = "65"
left = "260"
width = "40"
fontSize = "30"
hoverFontStyle = "Bold"
hoverForegroundColor = "red"
j ustification = "Center"
value = "+

cursor = "hand"
toolTip = "increase volume"
onClick = "player.settings.volume = player.settings.volume + 5"
 />
<TEXT
top = "155"
width = "300"
height = "30"
fontFace = "System"
backgroundColor = "blue"
fore groundColor = "white"
j ustification = "Center"
scrolling = "true"
scrollingAmount = "1"
scrollingDelay = "50"
value = "wmpprop:player.status"
 />
</VIEW>
</THEME>

Requirements

Windows Media Player version 7.0 or later.

See Also

- TEXT Element
- TEXT.toolTip
- TEXT.wordWrap
TEXT.wordWrap

The wordWrap attribute specifies or retrieves a value indicating whether word wrap is enabled or disabled.

Syntax

elementID.wordWrap

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Word wrap is enabled.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Word wrap is disabled. If the text does not fit within the control, the text is cropped.</td>
</tr>
</tbody>
</table>

Remarks

The Text control does not split words apart. If a word extends beyond the width of the control, word wrap is employed to move the word to the next line. If a single word is longer than the control width, that word is clipped and occupies a single line.

If the width attribute is not specified, wordWrap is ignored and the Text control will resize rather than wrapping the text.

If line breaks are desired in particular locations, they must be explicitly specified in value using "&\#13;" or "r". If the latter form is used when specifying the value directly, the string must be prefixed with "JScript:" and the actual value must be surrounded by single quotes, as shown below. This is not necessary if the value is set dynamically from within an event handler.

If wordWrap is set to true and toolTip is not specified, the ToolTip will display the full text of the value attribute. If no ToolTip is desired, set toolTip to "" (empty string).

Example

<THEME>
  <VIEW>
<TEXT
  width = "50"
  height = "200"
  wordWrap = "true"
  backgroundColor = "blue"
  foregroundColor = "white"
  value = "This is a test.\n\nIt is only a test."
/>
<TEXT
  width = "50"
  height = "200"
  left = "100"
  wordWrap = "true"
  backgroundColor = "green"
  foregroundColor = "white"
  value = "JScript:'This is a test.\n\nIt is only a test.'"
/>
</VIEW>
</THEME>

Requirements

Windows Media Player version 7.0 or later.

See Also

- AmbientAttributes.width
- TEXT Element
- TEXT.toolTip
- TEXT.value

Remarks

This will create a TEXT element that will display the current time of the media being played. All properties of this TEXT element can be overridden by explicitly specifying them.
Requirements

Windows Media Player 7.0 or later.

See Also

• TEXT Element

DURATION

This is a predefined TEXT element with the following default values.

value="wmprop:player.currentMedia.DurationString"
tabstop="true"
justification="right"

Remarks

This will create a TEXT element that will display the duration of the media being played. All properties of this TEXT element can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

• TEXT Element

© 2000-2003 Microsoft Corporation. All rights reserved.
STATUSTEXT

This is a predefined TEXT element with the following default values.

value="wmpprop:player.status"
tabstop="true"

Remarks

This will create a TEXT element that will display the playback status. All properties of this TEXT element can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

• TEXT Element

TRACKNAMETEXT

This is a predefined TEXT element with the following default values.

value="wmpprop:player.currentMedia.name"
tabstop="true"

Remarks

This will create a TEXT element that will display the name of the current media. All properties of this TEXT element can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also
THEME Element

The THEME element is the parent-level element of a skin, and contains one or more VIEW elements, which in turn contain all other elements within a skin. Within script code, the THEME element is accessed through the theme global attribute rather than through a name specified by an id attribute, which is not supported by the THEME element.

The THEME element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>Specifies or retrieves the name of the author of the skin.</td>
</tr>
<tr>
<td>authorVersion</td>
<td>Specifies or retrieves the version number of the skin as assigned by the author.</td>
</tr>
<tr>
<td>copyright</td>
<td>Specifies or retrieves the copyright string for the skin.</td>
</tr>
<tr>
<td>currentViewID</td>
<td>Specifies or retrieves the currently displayed VIEW.</td>
</tr>
<tr>
<td>title</td>
<td>Specifies or retrieves the title of the skin.</td>
</tr>
<tr>
<td>version</td>
<td>Specifies or retrieves the Windows Media Player version number for which the skin was authored. Can only be set at design time.</td>
</tr>
</tbody>
</table>

The THEME element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>closeView</td>
<td>Closes an open VIEW.</td>
</tr>
<tr>
<td>loadPreference</td>
<td>Loads a preference from the registry.</td>
</tr>
<tr>
<td>logString</td>
<td>Logs a user-defined string to the error file, if logging is enabled.</td>
</tr>
<tr>
<td>openDialog</td>
<td>Opens a file dialog box.</td>
</tr>
<tr>
<td>openView</td>
<td>Opens a VIEW in a new window.</td>
</tr>
<tr>
<td>openViewRelative</td>
<td>Opens a VIEW in a new window at a specified initial position relative to</td>
</tr>
</tbody>
</table>
The THEME element does not support event handlers.

See Also

- Skin Programming Reference

THEME.author

The **author** attribute specifies or retrieves the name of the author of the skin.

Syntax

```plaintext
theme.author
```

Possible Values

This attribute is a read/write **String** with no default value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME.Element
- THEME.authorVersion

© 2000-2003 Microsoft Corporation. All rights reserved.

THEME.author

The **author** attribute specifies or retrieves the name of the author of the skin.

Syntax

```plaintext
theme.author
```

Possible Values

This attribute is a read/write **String** with no default value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME.Element
- THEME.authorVersion
THEME.authorVersion

The **authorVersion** attribute specifies or retrieves the version number of the skin as assigned by the author.

**Syntax**

```plaintext
theme.authorVersion
```

**Possible Values**

This attribute is a read-only **String** with no default value.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- THEME Element
- THEME.author

---

THEME.closeView

The **closeView** method closes an open **VIEW**.

**Syntax**

```plaintext
theme.closeView(theView)
```

**Parameters**

- `theView`
A String specifying the id of the VIEW to close.

Return Values

This method does not return a value.

Example Code

```html
<THEME>
  <VIEW>
    <TEXT value="open"
        onclick="javascript:theme.openView('newView')"/>
    <TEXT top="30" value="close"
        onclick="javascript:theme.closeView('newView')"/>
  </VIEW>
  <VIEW id="newView"/>
</THEME>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- **THEME Element**
- **THEME.openView**

THEME.copyright

The `copyright` attribute specifies or retrieves the copyright string for the skin.

Syntax

```
theme.copyright
```

Possible Values

This attribute is a read/write String with no default value.

Requirements
The `currentViewID` attribute specifies or retrieves the currently displayed `VIEW`.

Syntax

```
theme.currentViewID
```

Possible Values

This attribute is a read/write `String` specifying the `id` of the current `VIEW`. It has no default value.

Example

```
<THEME currentViewID="startView">
  <VIEW>
    <TEXT value="this would have been the default view"/>
  </VIEW>
  <VIEW id="startView">
    <TEXT value="go to new view"
      onclick="javascript:theme.currentViewID='newView'"/>
  </VIEW>
  <VIEW id="newView">
    <TEXT value="new view"
    </VIEW>
</THEME>
```

Remarks

Specifying `currentViewID` automatically closes the existing `currentView` (pointed to by the `view` global attribute) and opens the specified `VIEW`.

Requirements

Windows Media Player version 7.0 or later.
THEME.loadPreference

The `loadPreference` method loads a preference from the registry.

Syntax

```javascript
theme.logString(theKey)
```

Parameters

`theKey`

A `String` specifying the key of the preference value to load.

Return Values

This method returns a `String`.

Remarks

A preference is a key/value pair that can be stored in the registry in order to retain information about the state of the Windows Media Player between runs. This feature can be used, for example, to save customization settings so that they won't have to be re-entered each time Windows Media Player is started.

Preferences are not encrypted and therefore are not a secure method for persisting data. Do not use preferences to store private data.

Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME Element
- THEME.savePreference
THEME.logString

The logString method logs a user-defined string to the error file, if logging is enabled.

Syntax

theme.logString(message)

Parameters

message

A String specifying the message to log.

Return Values

This method does not return a value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME Element

THEME.openDialog
The `openDialog` method opens a file dialog box.

**Syntax**

```
theme.openDialog(dialogType, parameters)
```

**Parameters**

- `dialogType`
  A `String` that specifies the type of dialog box. Must be set to "FILE_OPEN".

- `parameters`
  A `String` that can be used for additional information. Must be set to "FILES_ALLMEDIA".

**Return Values**

This method returns a `String` containing the URL of the selected file or "" (empty string) if the user clicks cancel.

**Remarks**

This method can be used for files on the local hard drives or on network drives.

**Example Code**

```xml
<THEME>
  <VIEW id="View1" backgroundImage="greenstone.bmp" width=500 height=300 author="Microsoft Corp.">
    <BUTTON id="Open" image="Open.png" onclick="jscript:
      player.URL = theme.openDialog('FILE_OPEN','FILES_ALLMEDIA')">
    </BUTTON>
  </VIEW>
</THEME>
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [THEME Element](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
THEME.openView

The openView method opens a VIEW in a new window.

Syntax

theme.openView(view)

Parameters

view

A String specifying the id of the VIEW to open.

Return Values

This method does not return a value.

Example Code

<THEME>
  <VIEW>
    <TEXT value="open"
      onclick="jscript:theme.openView('newView')"/>
    <TEXT top="30" value="close"
      onclick="jscript:theme.closeView('newView')"/>
  </VIEW>
  <VIEW id="newView"/>
</THEME>

Requirements

Windows Media Player version 7.0 or later.

See Also

- **THEME Element**
- **THEME.closeView**
- **THEME.openViewRelative**
THEME.openViewRelative

The `openViewRelative` method opens a `VIEW` in a new window at a specified initial position relative to the upper-left corner of the skin.

Syntax

```javascript
theme.openView(view, left, top)
```

Parameters

- `view`
  
  A `String` specifying the id of the `VIEW` to open.

- `left`
  
  A `Number (long)` specifying the initial distance in pixels of the left border of the `VIEW` from the left border of the skin. A negative value indicates an initial position to the left of the skin border.

- `top`
  
  A `Number (long)` specifying the initial position of the top border of the `VIEW` relative to the top border of the skin. A negative values indicates an initial position above the skin border.

Return Values

This method does not return a value.

Remarks

The position specified for the `VIEW` is used the first time this method is called, after which the user can drag the `VIEW` to another location. The new position is saved, and on subsequent calls, the most recent position is used.

Example Code

```xml
<THEME>
  <VIEW>
    <TEXT value="open"
      onclick="jscript:theme.openViewRelative('newView', 50, 50)"
      top="30" value="close"
      onclick="jscript:theme.closeView('newView')"/>
  </VIEW>
  <VIEW id="newView"/>
</THEME>
```
Requirements

Windows Media Player 9 Series or later.

See Also

- `THEME.Element`
- `THEME.closeView`
- `THEME.openView`

THEME.playSound

The `playSound` method plays the specified sound file.

Syntax

```
theme.playSound(soundFile)
```

Parameters

- `soundFile`

A `String` specifying the name of the sound file to play.

Return Values

This method does not return a value.

Remarks

This method allows you to add sound effects to a skin—for example, when buttons are clicked. The sound is played by the operating system directly and not by Windows Media Player. This means that the sound cannot be controlled with Windows Media Player settings and methods, but it can be played while Windows Media Player is playing another digital media file.

This method supports WAV files only.

Requirements
THEME.savePreference

The `savePreference` method saves a preference in the registry.

Syntax

```javascript
theme.savePreference(theKey, theValue)
```

Parameters

- **theKey**
  
  A `String` specifying the key of the preference value to save.

- **theValue**
  
  A `String` specifying the value to save.

Return Values

This method does not return a value.

Remarks

A preference is a key/value pair that can be stored in the registry in order to retain information about the state of Windows Media Player between runs. This feature can be used, for example, to save customization settings so that they won't have to be re-entered each time Windows Media Player is started.

Preferences are not encrypted and therefore are not a secure method for persisting data. Do not use preferences to store private data.

Example Code
Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME Element
- THEME.loadPreference

**THEME.showErrorDialog**

The `showErrorDialog` method displays the standard error dialog box.

**Syntax**

```javascript
theme.showErrorDialog()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Remarks**

If `Settings.enableErrorDialogs` is false, this method can be used to programmatically display the error dialog. If there are no errors in the error queue, the error dialog box is not shown.

For Windows Media Player 9 Series or later, this method must be called from the error event handler. Windows Media Player 9 Series clears the error queue for skins after the error event has been fired.
THEME.title

The title attribute specifies or retrieves the title of the skin.

Syntax

theme.title

Possible Values

This attribute is a read/write String with no default value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- THEME Element
  - Settings.enableErrorDialogs
THEME.version

The **version** attribute specifies or retrieves the Windows Media Player version number for which the skin was authored.

**Syntax**

```plaintext
theme.version
```

**Possible Values**

This attribute is a **Number** *(float)*. Specifying a value for this attribute at design time or run time has no effect. The value is always 1.0.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [THEME Element](#)

---

VIDEO Element

The **VIDEO** element provides a way to manipulate a video window in a skin, using the following attributes and events. A predefined **VIDEO** element is also provided for convenience.

The **VIDEO** element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code></td>
<td>Specifies or retrieves the background color of the Video control.</td>
</tr>
<tr>
<td><code>cursor</code></td>
<td>Specifies or retrieves the cursor value that is used when the mouse is over a clickable area of the video.</td>
</tr>
<tr>
<td><code>fullScreen</code></td>
<td>Specifies or retrieves a value indicating whether the video is displayed in full-screen mode. Can only be set at run time.</td>
</tr>
</tbody>
</table>
**maintainAspectRatio**
Specifies or retrieves a value indicating whether the video will maintain the aspect ratio when trying to fit within the width and height defined for the control.

**shrinkToFit**
Specifies or retrieves a value indicating whether the video will shrink to the width and height defined for the Video control.

**stretchToFit**
Specifies or retrieves a value indicating whether the video will stretch itself to the width and height defined for the Video control.

**tooltip**
Specifies or retrieves the ToolTip text for the video window.

**windowless**
Specifies or retrieves a value indicating whether the Video control will be windowed or windowless; that is, whether the entire rectangle of the control will be visible at all times or can be clipped. Can only be set at design time.

**zoom**
Specifies the percentage by which to scale the video.

The **VIDEO** element can implement the following event handlers.

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>onvideoend</strong></td>
<td>Handles an event that occurs when the video stops rendering and is unloaded.</td>
</tr>
<tr>
<td><strong>onvideostart</strong></td>
<td>Handles an event that occurs when the video is loaded and begins to render.</td>
</tr>
</tbody>
</table>

The **VIDEO** element supports the ambient attributes and can implement the ambient event handlers, except where noted. For more information, see Ambient Attributes and Ambient Event Handlers.

Predefined video elements are normal **VIDEO** elements with various common attribute settings specified by default. The following predefined video elements are available.

<table>
<thead>
<tr>
<th>Predefined <strong>VIDEO</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WMPVIDEO</strong></td>
<td>A <strong>VIDEO</strong> element that stretches the video when resized.</td>
</tr>
</tbody>
</table>

See Also

- Skin Programming Reference
**VIDEO.backgroundColor**

The `backgroundColor` attribute specifies or retrieves the background color of the Video control.

**Syntax**

```html
elementID.backgroundColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value or the value "none". It has a default value of "none", which means that if there is no video associated with the video control, the Video control is transparent and the background shows through.

**Remarks**

When the video is smaller than the window and `stretchToFit` is false, the background color appears around the video.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- [VIDEO Element](#)
- [VIDEO/stretchToFit](#)

---

**VIDEO.cursor**

The `cursor` attribute specifies or retrieves the cursor value that is used when the mouse is over a clickable area of the video.
Syntax

`elementID.cursor`

**Possible Values**

This attribute is a read/write **String**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Platform-dependent cursor (usually an arrow).</td>
</tr>
<tr>
<td>hand</td>
<td>Hand.</td>
</tr>
<tr>
<td>help</td>
<td>Arrow with question mark indicating Help is available.</td>
</tr>
<tr>
<td>sizeall</td>
<td>Four-pointed arrow pointing north, south, east, and west.</td>
</tr>
<tr>
<td>sizenesw</td>
<td>Double-pointed arrow pointing northeast and southwest.</td>
</tr>
<tr>
<td>sizens</td>
<td>Double-pointed arrow pointing north and south.</td>
</tr>
<tr>
<td>sizenwse</td>
<td>Double-pointed arrow pointing northwest and southeast.</td>
</tr>
<tr>
<td>sizewe</td>
<td>Double-pointed arrow pointing west and east.</td>
</tr>
<tr>
<td>uparrow</td>
<td>Vertical arrow pointing upward.</td>
</tr>
<tr>
<td>*.ani</td>
<td>Any .ani or .cur file (must be in the same directory as the .wms file or in the .wmz file).</td>
</tr>
<tr>
<td>*.cur</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**

For rendering purposes, system is the default cursor. The default value retrieved from this attribute is "" (empty string).

**Requirements**

Windows Media Player version 7.0 or later.

See Also

- **VIDEO Element**

© 2000-2003 Microsoft Corporation. All rights reserved.
VIDEO.fullScreen

The **fullScreen** attribute specifies or retrieves a value indicating whether the video is displayed in full-screen mode.

**Syntax**

```html
elementID.fullScreen
```

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Video displays in full-screen mode.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Video does not display in full-screen mode.</td>
</tr>
</tbody>
</table>

**Remarks**

This property can be specified only at run time, after a file has been loaded. It must therefore be set within a script event handler. The escape button is used to return to normal viewing.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [VIDEO Element](#)
- [VIDEO.maintainAspectRatio](#)
- [VIDEO.shrinkToFit](#)
- [VIDEO.stretchToFit](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**VIDEO.maintainAspectRatio**

The **maintainAspectRatio** attribute specifies or retrieves a value indicating whether the video will maintain its...
aspect ratio when trying to fit within the width and height defined for the control.

Syntax

elementID.maintainAspectRatio

Possible Values

This attribute is a read/write Boolean.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. Video maintains its aspect ratio when resizing.</td>
</tr>
<tr>
<td>false</td>
<td>Video does not maintain its aspect ratio when resizing.</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIDEO Element

VIDEO.onvideoend

The onvideoend event handler handles an event that occurs when the video stops rendering and is unloaded.

Syntax

onvideoend

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIDEO Element
The **onvideostart** event handler handles an event that occurs when the video is loaded and begins to render.

### Syntax

```javascript
onvideostart
```

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- [VIDEO Element](#)

---

### VIDEO.shrinkToFit

The **shrinkToFit** attribute specifies or retrieves a value indicating whether the video will shrink to the width and height defined for the Video control.

### Syntax

```javascript
elementID.shrinkToFit
```

### Possible Values
This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The video will shrink to fit the control.</td>
</tr>
<tr>
<td>false</td>
<td>The video will not shrink to fit the control.</td>
</tr>
</tbody>
</table>

**Remarks**

If no width or height is specified, this attribute is ignored.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [VIDEO Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**VIDEO.stretchToFit**

The **stretchToFit** attribute specifies a value indicating whether the video will stretch to the width and height defined for the Video control.

**Syntax**

`elementID.stretchToFit`

**Possible Values**

This attribute is a read/write **Boolean**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The video will stretch to fit the control.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The video will not stretch to fit the control.</td>
</tr>
</tbody>
</table>
VIDEO.toolTip

The `toolTip` attribute specifies or retrieves the ToolTip text for the video window.

Syntax

```
elementID.toolTip
```

Possible Values

This attribute is a read/write `String`, which must not exceed 1024 characters in length. It has no default value.

Remarks

When this attribute is set to "" (empty string), no ToolTip is displayed.

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIDEO Element
The **windowless** attribute specifies or retrieves a value indicating whether the Video control will be windowed or windowless; that is, whether the entire rectangle of the control will be visible at all times or can be clipped. Can only be set at design time.

**Syntax**

`elementID.windowless`

**Possible Values**

This attribute is a **Boolean** specified at design time and read-only thereafter.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Video control will be windowless.</td>
</tr>
<tr>
<td>false</td>
<td>Default. Video control will be windowed.</td>
</tr>
</tbody>
</table>

**Remarks**

If a non-rectangular video window is desired, or if you want to cover any part of the video window with an image, this attribute must be set to true. This sacrifices some performance in order to do the necessary clipping.

Video playback is optimized for unclipped playback. In this case, the **windowless** attribute is set to false, and the entire video rectangle is always displayed. Any image covering the video window is ignored, and the video window has the highest-level z-order.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **VIDEO Element**
VIDEO.zoom

The zoom attribute specifies the percentage by which to scale the video.

Syntax

elementID.zoom

Possible Values

This attribute is a read/write Number (long) ranging from 1 to the maximum size accommodated by the width and height of the Video control. It has a default value of 100.

Remarks

This attribute cannot be used in conjunction with the fullScreen attribute.

If width and height are specified, and the resulting video window is larger than the video being played, the video can be enlarged by scaling up to the maximum size accommodated by the window. If width and height are not specified, zoom is limited to values of 100 or less.

If the shrinkToFit property is false, the video will change proportion upon zooming in order to fit itself to the available space. If shrinkToFit is true, the video will shrink to fit within the most restrictive dimension, while retaining its original proportions.

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIDEO Element
- VIDEO.fullScreen
- VIDEO.shrinkToFit

© 2000-2003 Microsoft Corporation. All rights reserved.
WMPVIDEO

This is a predefined VIDEO element with the following default values.

```
backgroundColor="black"
horizontalAlignment="stretch"
verticalAlignment="stretch"
```

Remarks

This will create a VIDEO element that will stretch the video window when the skin is resized. Digital video displayed in this window will stretch to fit the available space or will be scaled up or down according to the settings on the Player View menu under Video Size. The View menu is displayed in the full mode of the Player or when the VIEW.titleBar attribute is set to true in a skin.

All properties of this VIDEO element can be overridden by explicitly specifying them.

Requirements

Windows Media Player 7.0 or later.

See Also

- [VIDEO Element](#)

VIDEOSETTINGS Element

The VIDEOSETTINGS element provides a way to modify various video settings, using the attributes and method listed here.

The VIDEOSETTINGS element supports the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brightness</td>
<td>Specifies or retrieves the brightness setting of the video.</td>
</tr>
<tr>
<td>contrast</td>
<td>Specifies or retrieves the contrast setting of the video.</td>
</tr>
</tbody>
</table>
The **VIDEOSETTINGS** element supports the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>Resets all attributes to their default values.</td>
</tr>
</tbody>
</table>

The **VIDEOSETTINGS** element can implement the `attribute_onchange` event handlers.

**Note**  This element requires Windows Media Player for Windows XP or later.

**See Also**

- [Skin Programming Reference](#)
VIDEOSETTINGS

The `contrast` attribute specifies or retrieves the contrast setting of the video.

Syntax

`elementID.contrast`

Possible Values

This attribute is a read/write `Number (long)` with a value ranging from -127 to +127 and a default value of 0.

Requirements

Windows Media Player version for Windows XP or later.

See Also

- `VIDEOSETTINGS Element`

VIDEOSETTINGS

The `hue` attribute specifies or retrieves the hue setting of the video.

Syntax

`elementID.hue`
Possible Values

This attribute is a read/write **Number (long)** with a value ranging from -127 to +127 and a default value of 0.

Requirements

Windows Media Player version for Windows XP or later.

See Also

- **VIDEOSETTINGS Element**

VIDE OSETTINGS.saturation

The **saturation** attribute specifies or retrieves the saturation setting of the video.

Syntax

```
elementID.saturation
```

Possible Values

This attribute is a read/write **Number (long)** with a value ranging from -127 to +127 and a default value of 0.

Requirements

Windows Media Player version for Windows XP or later.

See Also

- **VIDEOSETTINGS Element**
VIDEOSETTINGS.reset

The `reset` method resets all attributes to their default values of zero.

Syntax

```
elementID.reset()
```

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Requirements

Windows Media Player version for Windows XP or later.

See Also

- [VIDEOSETTINGS Element](#)

VIEW Element

The `VIEW` element contains the user interface details of a skin, and uses the attributes, methods, and event handlers shown in the following tables.

Multiple `VIEW` elements can be defined as children of the `THEME` element of a skin to provide different interfaces for different situations. `VIEW` elements cannot be specified as children of any other element, and they contain all other skin elements. Note that each view has its own variable scope, which means it cannot share attribute values with other views.
The view global attribute can be used to reference a specific VIEW element from anywhere within it. This is an alternative to using its id attribute, which must be used from within other VIEW elements or from within the THEME element.

The VIEW element supports the following attributes. Attributes marked with an asterisk (*) are also supported by the SUBVIEW element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor *</td>
<td>Specifies or retrieves the background color of the VIEW or SUBVIEW.</td>
</tr>
<tr>
<td>backgroundImage *</td>
<td>Specifies or retrieves the background image of the VIEW or SUBVIEW.</td>
</tr>
<tr>
<td>backgroundImageHueShift</td>
<td>Specifies or retrieves the amount by which the hue of the background image is shifted.</td>
</tr>
<tr>
<td>backgroundImageSaturation</td>
<td>Specifies or retrieves the saturation value of the background image.</td>
</tr>
<tr>
<td>backgroundTiled *</td>
<td>Specifies or retrieves a value indicating whether the background image of the VIEW or SUBVIEW is tiled.</td>
</tr>
<tr>
<td>category</td>
<td>Specifies or retrieves the category for which the user interface will appear.</td>
</tr>
<tr>
<td>focusObjectID</td>
<td>Specifies or retrieves a value indicating which element has keyboard focus.</td>
</tr>
<tr>
<td>maxHeight</td>
<td>Specifies or retrieves the maximum height of the VIEW when resizing.</td>
</tr>
<tr>
<td>maxWidth</td>
<td>Specifies or retrieves the maximum width of the VIEW when resizing.</td>
</tr>
<tr>
<td>minHeight</td>
<td>Specifies or retrieves the minimum height of the VIEW when resizing.</td>
</tr>
<tr>
<td>minWidth</td>
<td>Specifies or retrieves the minimum width of the VIEW when resizing.</td>
</tr>
<tr>
<td>resizable</td>
<td>Specifies or retrieves a value indicating whether the VIEW can be resized.</td>
</tr>
<tr>
<td>resizeBackgroundImage</td>
<td>Specifies or retrieves a value indicating whether the background image can be resized.</td>
</tr>
<tr>
<td>scriptFile</td>
<td>Specifies the file names of accompanying JScript files.</td>
</tr>
<tr>
<td>timerInterval</td>
<td>Specifies or retrieves the interval, in milliseconds, at which the timer fires events to the ontimer event handler.</td>
</tr>
<tr>
<td>title</td>
<td>Specifies or retrieves the title of the VIEW. Can only be set at design time.</td>
</tr>
<tr>
<td>titleBar</td>
<td>Specifies or retrieves a value indicating whether the window title bar is shown.</td>
</tr>
<tr>
<td>transparencyColor *</td>
<td>Specifies or retrieves the transparency color of the background.</td>
</tr>
</tbody>
</table>
The **VIEW** element supports the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>Closes the <strong>VIEW</strong>.</td>
</tr>
<tr>
<td>maximize</td>
<td>Maximizes the <strong>VIEW</strong>.</td>
</tr>
<tr>
<td>minimize</td>
<td>Minimizes the <strong>VIEW</strong>.</td>
</tr>
<tr>
<td>restore</td>
<td>Restores the <strong>VIEW</strong>.</td>
</tr>
<tr>
<td>returnToMediaCenter</td>
<td>Returns the user to the full mode of Windows Media Player.</td>
</tr>
<tr>
<td>size</td>
<td>Resizes the <strong>VIEW</strong> on a specified edge.</td>
</tr>
</tbody>
</table>

The **VIEW** element can implement the following event handlers.

<table>
<thead>
<tr>
<th>Event handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onclose</td>
<td>Handles an event that occurs when the <strong>VIEW</strong> is about to be closed.</td>
</tr>
<tr>
<td>onerror</td>
<td>Handles an error event if <code>Settings.enableErrorDialogs</code> is set to false.</td>
</tr>
<tr>
<td>onload</td>
<td>Handles an event that occurs when the <strong>VIEW</strong> is first displayed.</td>
</tr>
<tr>
<td>ontimer</td>
<td>Handles timer events.</td>
</tr>
</tbody>
</table>

The **VIEW** element supports the ambient attributes and can implement the ambient event handlers, except where noted. For more information, see [Ambient Attributes](#) and [Ambient Event Handlers](#).

**See Also**

- [Skin Programming Reference](#)
The **backgroundColor** attribute specifies or retrieves the background color of the **VIEW** or **SUBVIEW**.

**Syntax**

```plaintext
elementID.backgroundColor
```

**Possible Values**

This attribute is a read/write **String** containing any Microsoft Internet Explorer color value or the value "none". It has a default value of "white" for **VIEW** elements or "none" for **SUBVIEW** elements.

In a Windows Media Download (WMD) package, if you specify the **backgroundColor** attribute for a **VIEW** element, then you must also specify the **backgroundColor** attribute for that element.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [Color Reference](#)
- [**VIEW** Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**VIEW**.**backgroundImage**

The **backgroundImage** attribute specifies or retrieves the background image of the **VIEW** or **SUBVIEW**.

**Syntax**

```plaintext
elementID.backgroundImage
```

**Possible Values**

This attribute is a read/write **String**.

**Remarks**

The supported formats are BMP, JPG, GIF, and PNG. If the image is an 8-bit BMP file, its hue and saturation values can be changed dynamically using the **backgroundImageHueShift** and **backgroundImageSaturation** attributes.
In a Windows Media Download (WMD) package, if you specify the `backgroundImage` attribute for a `VIEW` element, then you must also specify the `backgroundColor` attribute for that element.

Requirements

Windows Media Player version 7.0 or later.

See Also

- `VIEW Element`
- `VIEW.backgroundImageHueShift`
- `VIEW.backgroundImageSaturation`

---

**VIEW.backgroundImageHueShift**

The `backgroundImageHueShift` attribute specifies or retrieves the amount by which the hue of the background image is shifted.

**Syntax**

```
elementID.backgroundImageHueShift
```

**Possible Values**

This attribute is a read/write `Number (float)` with a value ranging from 0.0 to 360.0 with a default value of 0.0.

**Remarks**

This attribute changes the hue value of the images specified by the `backgroundImage` attribute if it has been specified and it refers to an 8-bit BMP image.

**Requirements**

Windows Media Player 9 Series or later.

See Also

- `VIEW Element`
- `VIEW.backgroundImage`
**VIEW.backgroundImageSaturation**

The **backgroundImageSaturation** attribute specifies or retrieves the saturation value of the background image.

**Syntax**

```
elementID.backgroundImageSaturation
```

**Possible Values**

This attribute is a read/write **Number (float)** with a value ranging from 0.0 to 2.0 with a default value of 1.0.

**Remarks**

This attribute changes the saturation value of the images specified by the **backgroundImage** attribute if it has been specified and it refers to an 8-bit BMP image.

**Requirements**

Windows Media Player 9 Series or later.

**See Also**

- **VIEW Element**
- **VIEW.backgroundImage**
- **VIEW.backgroundImageHueShift**
VIEW.backgroundTiled

The `backgroundTiled` attribute specifies or retrieves a value indicating whether the background image of the `VIEW` or `SUBVIEW` is tiled.

Syntax

```
elementID.backgroundTiled
```

Possible Values

This attribute is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The image is repeated horizontally and vertically.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The image is not repeated.</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player version 7.0 or later.

See Also

- `VIEW Element`

VIEW.category

The `category` attribute specifies or retrieves the category for which the `VIEW` is intended.

Syntax

```
elementID.category
```

Possible Values
This attribute is a read/write String containing one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Default. Theme for all types of media</td>
</tr>
<tr>
<td>Radio</td>
<td>The UI for Radio Playback</td>
</tr>
<tr>
<td>CD</td>
<td>The UI for CD playback</td>
</tr>
<tr>
<td>DVD</td>
<td>The UI for DVD playback</td>
</tr>
<tr>
<td>Music</td>
<td>The UI for MP3, WAV, MIDI, WMA</td>
</tr>
<tr>
<td>Video</td>
<td>The UI for Video playback</td>
</tr>
</tbody>
</table>

Requirements

Windows Media Player version 7.0 or later.

See Also

- [VIEW Element](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

VIEW.close

The close method closes the VIEW.

**Syntax**

```
elementID.close()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Requirements**
VIEW.focusObjectID

The focusObjectID attribute specifies or retrieves a value indicating which element has keyboard focus. Can only be set at run time.

Syntax

`elementID.focusObjectID`

Possible Values

This attribute is a read/write String specifying the id of the element that has focus. It has no default value.

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element
VIEW.maxHeight

The `maxHeight` attribute specifies or retrieves the maximum height in pixels of the `<VIEW>` when resizing.

**Syntax**

`elementID.maxHeight`

**Possible Values**

This attribute is a read/write `Number (long)` with a value of zero or greater. It has a default value of zero, which means there is no restriction on the maximum height of the `<VIEW>`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `<VIEW>` Element
- `<VIEW.minHeight>`

---

VIEW.maximize

The `maximize` method maximizes the `<VIEW>`.

**Syntax**

`elementID.maximize()`

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

---

© 2000-2003 Microsoft Corporation. All rights reserved.
**Example**

```xml
<THEME>
    <VIEW id="View1">
        <BUTTON id="Open" Image="Open.png" onclick="javascript:View1.maximize()">
        </BUTTON>
    </VIEW>
</THEME>
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- VIEW Element
- VIEW.minimize

---

**VIEW.maxWidth**

The `maxWidth` attribute specifies or retrieves the maximum width in pixels of the VIEW when resizing.

**Syntax**

```xml
elementID.maxWidth
```

**Possible Values**

This attribute is a read/write Number (long) with a value of zero or greater. It has a default value of zero, which means there is no restriction on the maximum width of the VIEW.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- VIEW Element
- VIEW.minWidth
The `minHeight` attribute specifies or retrieves the minimum height in pixels of the `VIEW` when resizing.

**Syntax**

`elementID.minHeight`

**Possible Values**

This attribute is a read/write `Number (long)` with a value of zero or greater. It has a default value of zero, which means there is no restriction on the minimum height of the `VIEW`.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `VIEW Element`
- `VIEW.maxHeight`

---

The `minimize` method minimizes the `VIEW`.

**Syntax**
elementID.minimize()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Example

<THEME>
  <VIEW id="View1">
    <BUTTON id="Open" image="Open.png"
      onclick="javascript:View1.minimize()">
    </BUTTON>
  </VIEW>
</THEME>

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element
- VIEW.maximize

VIEW.minWidth

The minWidth attribute specifies or retrieves the minimum width in pixels of the VIEW when resizing.

Syntax

elementID.minWidth

Possible Values

This attribute is a read/write Number (long) with a value of zero or greater. It has a default value of zero, which means there is no restriction on the minimum width of the VIEW.
Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element
- VIEW.maxWidth

© 2000-2003 Microsoft Corporation. All rights reserved.

VIEW.onerror

The onerror event handler handles an event that occurs when the VIEW is about to be closed.

Syntax

onerror

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element

© 2000-2003 Microsoft Corporation. All rights reserved.

VIEW.onerror

© 2000-2003 Microsoft Corporation. All rights reserved.
The `onerror` event handles an error event if `Settings.enableErrorDialogs` is set to false.

Syntax

`onerror`

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element

---

The `onload` event handler handles an event that occurs when the VIEW is first displayed.

Syntax

`onload`

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW Element

---

© 2000-2003 Microsoft Corporation. All rights reserved.
VIEW.ontimer

The `ontimer` event handles timer events.

**Syntax**

`ontimer`

**Remarks**

The `timerInterval` attribute specifies the interval at which timer events will be fired.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `VIEW.Element`
- `VIEW.timerInterval`

---

VIEW.resizeable

The `resizeable` attribute retrieves a value indicating whether the VIEW can be resized.

**Syntax**

`elementID.resizeable`

**Possible Values**

This attribute is a read-only `Boolean` with a default value equal to the `titlebar` attribute.
The `resizeBackgroundImage` attribute specifies or retrieves a value indicating whether the background image can be resized.

**Syntax**

`elementID.resizeBackgroundImage`

**Possible Values**

This attribute is a read/write `Boolean`.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The background image can be resized.</td>
</tr>
<tr>
<td>false</td>
<td>Default. The background image cannot be resized.</td>
</tr>
</tbody>
</table>

**Remarks**

If there is no `titlebar`, and therefore no window or border, you must use the `size` method to resize the `VIEW` element.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `VIEW Element`
If you set this attribute to true, the background image will resize to fit the current values of the width and height attributes.

Requirements

Windows Media Player 9 Series or later.

See Also

- VIEW Element
- VIEW.backgroundImage

VIEW.restore

The restore method restores the VIEW.

Syntax

elementID.restore()

Parameters

This method takes no parameters.

Return Values

This method does not return a value.

Example

<THEME>
  <VIEW id="View1">
    <BUTTON id="Open" image="Open.png"
        onclick="javascript:View1.restore()">
    </BUTTON>
  </VIEW>
</THEME>

Requirements

Windows Media Player version 7.0 or later.
VIEW.returnToMediaCenter

The **returnToMediaCenter** method returns the user to the full mode of Windows Media Player.

**Syntax**

```
ElementID.returnToMediaCenter()
```

**Parameters**

This method takes no parameters.

**Return Values**

This method does not return a value.

**Example**

```
<THEME>
  <VIEW id="View1" backgroundImage="greenstone.bmp">
    <BUTTON id="Open" image="Open.png"
      onclick="jscript:View1.returnToMediaCenter()">
      </BUTTON>
  </VIEW>
</THEME>
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- **VIEW Element**
The `scriptFile` attribute specifies the file names of accompanying JScript files.

### Syntax

```
elementID.scriptFile
```

### Possible Values

This attribute is a write-only `String` with no default value. The file names of multiple JScript files are delimited with semicolons. Leading and following spaces and semicolons should not be present.

### Example

```
<VIEW id="theView" scriptFile="theScript.js">
</VIEW>
```

### Remarks

The code in the specified files can be used by any event handler in the View. Code that is used by event handlers in multiple Views can be placed in a file with the same name as the skin definition file but with a `.js` extension instead of a `.wms` extension. This file is loaded automatically and need not be specified in the skin definition file.

### Requirements

Windows Media Player version 7.0 or later.

### See Also

- [VIEW Element](#)
The size method resizes the VIEW on a specified edge.

Syntax

`elementID.size(handle)`

Parameters

`handle`

String specifying the edge or corner to move when sizing. This string must have one of the following eight values.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>topright</td>
</tr>
<tr>
<td>right</td>
<td>bottomright</td>
</tr>
<tr>
<td>bottom</td>
<td>bottomleft</td>
</tr>
<tr>
<td>left</td>
<td>topleft</td>
</tr>
</tbody>
</table>

Return Values

This method does not return a value.

Example

```xml
<THEME>
  <VIEW id="View1" backgroundImage="greenstone.bmp">
    <BUTTON id="sizer" horizontalAlignment="right" verticalAlignment="bottom" image="Open.png" onmousedown="javascript:View1.size('bottomright')">
    </BUTTON>
  </VIEW>
</THEME>
```

Remarks

This method is typically called from within an onmousedown handler. It takes care of resizing while the mouse is dragged and stops resizing when the mouse button is released. If the size of the VIEW is restricted, you cannot drag the mouse to resize the View beyond the restricted bounds.

Requirements
VIEW.timerInterval

The `timerInterval` attribute specifies or retrieves the interval, in milliseconds, at which the timer fires events to the `ontimer` event handler.

Syntax

```html
elementID.timerInterval
```

Possible Values

This attribute is a read/write `Number (long)` with a default value of 1000.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The timer event does not fire.</td>
</tr>
<tr>
<td>50 and above</td>
<td>The interval in milliseconds.</td>
</tr>
</tbody>
</table>

Any value below 50 (including negative numbers, but not including zero) generates an error and the previous value is maintained.

Remarks

This will not fire automatically if no `ontimer` event handler is implemented. Thus there is no performance degradation even though the default value is non-zero.

Requirements

Windows Media Player version 7.0 or later.

See Also
VIEW.title

The title attribute specifies or retrieves the title of the VIEW.

Syntax

`elementID.title`

Possible Values

This attribute is a String specified at design time and read-only thereafter. It can be up to 255 characters long and has no default value.

Remarks

The title specified or retrieved is the display name for the theme.

Requirements

Windows Media Player version 7.0 or later.

See Also

- VIEW.Element
- VIEW.titleBar
VIEW.titleBar

The `titleBar` attribute retrieves a value indicating whether the window title bar is shown.

**Syntax**

```
elementID.titleBar
```

**Possible Values**

This attribute is a read-only `Boolean`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Default. The window title bar is shown.</td>
</tr>
<tr>
<td>false</td>
<td>The window title bar is not shown.</td>
</tr>
</tbody>
</table>

**Remarks**

If the title bar is shown, the control box, minimize, and close buttons will be shown. The title of the window will be the title of the `VIEW` element.

If `titleBar` is set to true and the user attempts to change the value of `Video.zoom`, the change will not take place unless the skin monitors `zoom` and takes appropriate action to resize itself.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- `VIEW Element`
- `VIEW.title`
- `VIDEO.zoom`
The `transparencyColor` attribute specifies or retrieves the transparency color of the background image.

**Syntax**

```html
elementID.transparencyColor
```

**Possible Values**

This attribute is a read/write `String` containing any Microsoft Internet Explorer color value. It has no default value.

Because JPGs are lossy and therefore subject to unexpected color change, they are not recommended when `transparencyColor` is used.

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- [VIEW Element](#)
- [Color Reference](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

## Miscellaneous

This section documents specialized attributes and other miscellaneous topics for use in creating skins.

The following topics are covered.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Reference</td>
<td>A chart of colors, color names, and hexadecimal values for colors supported by skins.</td>
</tr>
<tr>
<td>Global Attributes</td>
<td>Global attributes are attributes that provide easy access to certain player elements or objects from anywhere within a skin.</td>
</tr>
<tr>
<td>Listening Attributes</td>
<td>Attributes used to connect one attribute to another so that its value changes every time the value of the other attribute changes.</td>
</tr>
</tbody>
</table>
Color Reference

The following colors are supported for use with skins. Every attribute that takes a color value can be specified using one of the following color names or hexadecimal values. Any six-digit hexadecimal number can be used, but only the ones on this chart have valid names.

Some attributes will also accept a value of "none". For **transparencyColor** attributes, "none" means that no transparency is used. For other color attributes as appropriate, "none" means that transparency is used instead of a color.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Hexadecimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliceblue</td>
<td>#F0F8FF</td>
</tr>
<tr>
<td>antiquewhite</td>
<td>#FAEBD7</td>
</tr>
<tr>
<td>aqua</td>
<td>#00FFFF</td>
</tr>
<tr>
<td>aquamarine</td>
<td>#7FFFD4</td>
</tr>
<tr>
<td>azure</td>
<td>#00FFFF</td>
</tr>
<tr>
<td>beige</td>
<td>#F5F5DC</td>
</tr>
<tr>
<td>bisque</td>
<td>#FFE4C4</td>
</tr>
<tr>
<td>black</td>
<td>#000000</td>
</tr>
<tr>
<td>blanchedalmond</td>
<td>#FFEBBD</td>
</tr>
<tr>
<td>blue</td>
<td>#0000FF</td>
</tr>
<tr>
<td>blueviolet</td>
<td>#8A2BE2</td>
</tr>
<tr>
<td>brown</td>
<td>#A52A2A</td>
</tr>
<tr>
<td>burlywood</td>
<td>#DEB887</td>
</tr>
<tr>
<td>cadetblue</td>
<td>#5F9EA0</td>
</tr>
<tr>
<td>chartreuse</td>
<td>#7FFF00</td>
</tr>
<tr>
<td>chocolate</td>
<td>#D2691E</td>
</tr>
<tr>
<td>coral</td>
<td>#FF7500</td>
</tr>
<tr>
<td>cornflowerblue</td>
<td>#6495ED</td>
</tr>
<tr>
<td>cornsilk</td>
<td>#FFF8DC</td>
</tr>
<tr>
<td>crimson</td>
<td>#DC143C</td>
</tr>
<tr>
<td>cyan</td>
<td>#00FFFF</td>
</tr>
<tr>
<td>darkblue</td>
<td>#000008B</td>
</tr>
<tr>
<td>darkcyan</td>
<td>#008B8B</td>
</tr>
<tr>
<td>darkgoldenrod</td>
<td>#B8860B</td>
</tr>
<tr>
<td>darkgray</td>
<td>#A9A9A9</td>
</tr>
<tr>
<td>darkgreen</td>
<td>#006400</td>
</tr>
<tr>
<td>darkkhaki</td>
<td>#BDB76B</td>
</tr>
<tr>
<td>darkmagenta</td>
<td>#8B008B</td>
</tr>
<tr>
<td>darkolivegreen</td>
<td>#556B2F</td>
</tr>
<tr>
<td>darkorange</td>
<td>#FF8C00</td>
</tr>
<tr>
<td>darkorchid</td>
<td>#9932CC</td>
</tr>
<tr>
<td>darkred</td>
<td>#8B0000</td>
</tr>
<tr>
<td>darksalmon</td>
<td>#E9967A</td>
</tr>
<tr>
<td>darkseagreen</td>
<td>#8FBC8B</td>
</tr>
<tr>
<td>darkslateblue</td>
<td>#483D8B</td>
</tr>
<tr>
<td>darkslategray</td>
<td>#2F4F4F</td>
</tr>
<tr>
<td>darkturquoise</td>
<td>#00CED1</td>
</tr>
<tr>
<td>darkviolet</td>
<td>#9400D3</td>
</tr>
<tr>
<td>deepink</td>
<td>#FF1493</td>
</tr>
<tr>
<td>deepskyblue</td>
<td>#00BFFF</td>
</tr>
<tr>
<td>dimgray</td>
<td></td>
</tr>
<tr>
<td>dodgerblue</td>
<td></td>
</tr>
<tr>
<td>firebrick</td>
<td></td>
</tr>
<tr>
<td>floralwhite</td>
<td></td>
</tr>
<tr>
<td>#696969</td>
<td>#1E90FF</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>forestgreen</td>
<td>fuchsia</td>
</tr>
<tr>
<td>#228B22</td>
<td>#FF00FF</td>
</tr>
<tr>
<td>gold</td>
<td>goldenrod</td>
</tr>
<tr>
<td>#FFD700</td>
<td>#DAA520</td>
</tr>
<tr>
<td>greenyellow</td>
<td>honeydew</td>
</tr>
<tr>
<td>#ADFF2F</td>
<td>#FF00FF</td>
</tr>
<tr>
<td>indigo</td>
<td>ivory</td>
</tr>
<tr>
<td>#4B0082</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>lavenderblush</td>
<td>lawngreen</td>
</tr>
<tr>
<td>#FF00FF</td>
<td>#7CFC00</td>
</tr>
<tr>
<td>lightcoral</td>
<td>lightcyan</td>
</tr>
<tr>
<td>#F08080</td>
<td>#E0FFFF</td>
</tr>
<tr>
<td>lightgrey</td>
<td>lightpink</td>
</tr>
<tr>
<td>#808080</td>
<td>#FF69B4</td>
</tr>
<tr>
<td>lime</td>
<td>limegreen</td>
</tr>
<tr>
<td>#00FF00</td>
<td>#32CD32</td>
</tr>
<tr>
<td>maroon</td>
<td>mediumaquamarine</td>
</tr>
<tr>
<td>#800000</td>
<td>#66CDAA</td>
</tr>
<tr>
<td>mediumpurple</td>
<td>mediumseagreen</td>
</tr>
<tr>
<td>#9370DB</td>
<td>#3CB371</td>
</tr>
<tr>
<td>mediumturquoise</td>
<td>mediumvioletred</td>
</tr>
<tr>
<td>#48D1CC</td>
<td>#C71585</td>
</tr>
<tr>
<td>mistyrose</td>
<td>moccasin</td>
</tr>
<tr>
<td>#FFE4E1</td>
<td>#FFE4B5</td>
</tr>
<tr>
<td>oldlace</td>
<td>olive</td>
</tr>
<tr>
<td>#FDF5E6</td>
<td>#808000</td>
</tr>
<tr>
<td>orangered</td>
<td>orchid</td>
</tr>
<tr>
<td>#FF4500</td>
<td>#DA70D6</td>
</tr>
<tr>
<td>paleturquoise</td>
<td>palevioletred</td>
</tr>
<tr>
<td>#AFEEEE</td>
<td>#DB7093</td>
</tr>
<tr>
<td>peru</td>
<td>pink</td>
</tr>
<tr>
<td>#CD853F</td>
<td>#FFC0CB</td>
</tr>
<tr>
<td>purple</td>
<td>red</td>
</tr>
<tr>
<td>#800080</td>
<td>#FF0000</td>
</tr>
<tr>
<td>saddlebrown</td>
<td>salmon</td>
</tr>
<tr>
<td>#FB4513</td>
<td>#FA8072</td>
</tr>
<tr>
<td>seashell</td>
<td>sienna</td>
</tr>
<tr>
<td>#FF55EE</td>
<td>#A0522D</td>
</tr>
<tr>
<td>slateblue</td>
<td>slategray</td>
</tr>
<tr>
<td>#6A5ACD</td>
<td>#708090</td>
</tr>
<tr>
<td>steelblue</td>
<td>tan</td>
</tr>
</tbody>
</table>
Global Attributes

Global attributes are attributes that provide easy access to certain player elements or objects from anywhere within a skin.

The **player** global attribute is a reference to the **Player** object, and is used to access the primary functionality of Windows Media Player. The following example uses **player** to begin digital media playback.

```html
<BUTTON
   onclick="jscript:player.controls.play();"
/>```

The **theme** global attribute is a reference to the **THEME** element. This is the proper way to access **THEME** attributes, rather than by specifying an ID within the **THEME** element. The following example uses **theme** to open a new view.

```html
<TEXT
   value="open"
   onclick="jscript:theme.openView('newView');"
/>```

The **view** global attribute is a reference to the current **VIEW**. This can be used instead of the ID specified within the various **VIEW** elements. The following example uses **view** to close the current view.

```html
<BUTTON
   id="quitbutton"
   onclick="jscript:view.close();"
/>```
The **event** global attribute is used to access ambient event attributes from within event handlers. The following example uses **event** to determine whether the ALT key is pressed when a button is clicked.

```html<br>
<BUTTON onclick="javascript:if (event.altKey == true) myText.value='ALT';"
/><br>
```

The **playerApplication** global attribute is a reference to the **PlayerApplication** object, and is used by skin files provided as custom user interfaces for remoted Player controls. The Player control can be embedded in remote mode only in C++ programs that implement the **IWMPRemoteMediaServices** interface. The following example uses **playerApplication** to switch to the full mode of the Player.

```html<br>
<BUTTON onclick="javascript:playerApplication.switchToPlayerApplication();"
/><br>
```

For more information, see [Ambient Event Attributes](#).

**See Also**

- [Miscellaneous](#)

---

### Listening Attributes

A listening attribute is used to connect one attribute to another so that its value changes every time the value of the other attribute changes.

The listening attribute **wmpprop**: is the most useful. If the value of one attribute is specified to be the **wmpprop**: of a second attribute, the first value will be automatically updated to reflect the second value each time the second value changes.

**Example:**

```html<br>
<TEXT value="wmpprop:mySlider.value"
/><br>
```

In this way, the value of mySlider, shown by the position of the slider control, can also be shown as a number within a text box.
The two other listening attributes, `wmpenabled:` and `wmpdisabled:`, can be used to change the `enabled`
attribute of a control depending on whether its functionality is currently available in the player. These attributes
can listen only to methods of the `Controls` object.

Example:

```xml
<BUTTON
   id="play"
   enabled="wmpenabled:player.controls.play"
/>```

See Also

- Miscellaneous

---

**Windows Media Player Plug-ins**

Microsoft Windows Media Player exposes interfaces that allow you to extend functionality in several ways. The
following sections detail the supported plug-in architectures:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Windows Media Player Custom Visualizations</a></td>
<td>Windows Media Player visualizations are COM objects used to display visual imagery that is synchronized to the audio portion of the media playback of the player. Custom visualizations can be created using Visual C++.</td>
</tr>
<tr>
<td><a href="#">Windows Media Player User Interface Plug-ins</a></td>
<td>Windows Media Player user interface plug-ins add new functionality to the <strong>Now Playing</strong> pane of the full mode Player. You can create plug-ins that use the visualization area, a separate window, the settings area, the metadata area, or background plug-ins that expose no visible user interface.</td>
</tr>
</tbody>
</table>
Windows Media Player Custom Visualizations

Microsoft Windows Media Player provides visualizations that enable the user to see visual imagery that is synchronized to the sound of the media content as it plays. Several standard visualizations are included with Windows Media Player, including Spikes, Bars, and DotPlane. This section of the SDK provides programming information that will enable you to create your own visualizations.

The custom visualization documentation is divided into three sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About Custom Visualizations</strong></td>
<td>Discusses the architecture of custom visualizations in abstract terms. You should read this section to understand how visualizations relate to Windows Media Player.</td>
</tr>
<tr>
<td><strong>Custom Visualization Programming Guide</strong></td>
<td>Explains what you need to do to create a custom visualization. This section contains useful hints and tips that will help you create interesting visualizations.</td>
</tr>
<tr>
<td><strong>Custom Visualization Programming Reference</strong></td>
<td>Provides a reference for the interface you must implement as well as other technical reference information.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player Rendering Plug-ins

Windows Media Player rendering plug-ins decode (if necessary) and render custom data contained in a Windows Media format stream. Rendering plug-ins are DirectX Media Objects (DMO) that connect to the Player through COM interfaces.
About Custom Visualizations

Windows Media Player provides an interface to create custom visualizations. This section provides an overview of the architecture of visualizations, information about how they relate to Windows Media Player, and the technical details you will need to create custom visualizations.

Note that several places in the code, you will see the word Effect. Visualizations are effects.

Custom visualizations are described in greater detail in the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Overview</td>
<td>Describes visualizations from a user perspective.</td>
</tr>
<tr>
<td>Developer Overview</td>
<td>Describes visualizations from a developer perspective.</td>
</tr>
<tr>
<td>Building a Visualization</td>
<td>Describes how to create a visualization project.</td>
</tr>
<tr>
<td>Implementing Your Code</td>
<td>Describes how to modify the sample code generated by the visualization plug-in wizard.</td>
</tr>
<tr>
<td>Installing Visualizations</td>
<td>Describes how to install a visualization on a user's computer.</td>
</tr>
<tr>
<td>Submitting Your Visualization</td>
<td>Describes how to publicly distribute your completed visualization.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player Custom Visualizations
User Overview

To the user, visualizations are images that dance in time to the music. They provide additional entertainment to enhance musical experiences. The more entertaining a visualization is, the better!

Here is what a typical visualization looks like:

![Visualization Image]

**Standard Visualizations**

Windows Media Player includes several visualizations, such as:

- Ambience
- Plenoptic
- Spike

**Standard Presets**

In addition, each visualization comes with specific presets that alter how the visualization is displayed. For example, the Spike visualization comes in two presets:

- Spike
- Amoeba

Both effects are part of the same visualization, but have different presets. A preset is a set of values for the attributes of a particular visualization.

**Skin Attributes**

If you create skins for Windows Media Player, you can display the standard visualizations in your skins using the **Effects** element. For more details on using visualizations in skins, see [Windows Media Player Skins](#).

See Also

- [About Custom Visualizations](#)
Developer Overview

From the developer's point of view, visualizations are software programs that take audio data provided by Windows Media Player and convert that data to graphics that will please the eye of the user. The main subjects a developer needs to understand in order to create a new visualization are:

Visualization Packaging

Visualizations are COM controls that Windows Media Player uses to turn audio waveforms into animated graphics in Microsoft Windows. The COM controls are packaged as Microsoft Windows dynamic-link libraries (DLLs) and must be registered in the Windows registry. When Windows Media Player runs, registered custom visualizations are loaded and viewed in accordance with the instructions of the skin that Windows Media Player is using.

Audio Input

Windows Media Player provides your code with snapshots of audio frequency and waveform data at timed intervals measured in fractions of a second. The snapshot interval is internally determined by the Windows Media Player.

Graphical Output

The graphical output from your visualization is a Microsoft Windows device context. This is a standard Windows drawing surface that you can draw upon every time an audio snapshot is provided. All of the background Windows technology is taken care of for you. You just have to draw on the device context with the audio data provided.

Drawing Tools

You can draw on the device context with standard Microsoft Windows Graphics Device Interface (GDI) functions, using pens and brushes to create designs that are modified by the audio data supplied to you by Windows Media Player. GDI provides a rich set of drawing tools that can create many kinds of visual effects.

Programming Language

Microsoft Visual C++ 5.0 and higher is the only supported language for creating custom visualizations.

Plug-in Wizard

Windows Media Player provides a COM wizard that you can add to Visual C++ 6.0 that will generate the underlying code needed for your visualization. Not only are all source files provided, but a sample skin is generated to make it easy to test your visualization. The generated code creates a visualization similar to Bars, with two presets. You can then modify the code to create your own visualization. A registry file is also generated to register your visualization so that Windows Media Player can load it.

The following topic describes how visualization code processes audio data:

- Flow of Control

See Also
Flow of Control

Audio data comes into Windows Media Player continuously through a file or a stream. That data is passed to your visualization. You draw on a defined surface and pass that surface back to Windows Media Player. This interchange happens several times a second, and to the user, the result is a pleasing animation that moves in time to the music.

Here is the specific sequence of the visualization program flow:

1. At a timed interval, Windows Media Player takes a snapshot of the audio that is playing.
2. Windows Media Player supplies the data from that snapshot to your visualization through the `Render` function and the `RenderWindowed` function.
3. You must write code that will run when `Render` and `RenderWindowed` is called. Your code draws by using a device context defined by Windows Media Player when rendering windowless, or by using a window that you create when rendering windowed.
4. In a region specified by the current skin, Windows Media Player displays what your code drew.
5. This process repeats several times a second, creating graphical animations that are timed to the music. When the music stops playing, the visualization stops.

See Also

- [Developer Overview](#)
1. Microsoft Visual C++ 6.0, or Microsoft Visual C++ .NET. These are the only compilers supported. Included with Visual C++ is the Microsoft Windows Platform SDK which will provide detailed instructions on using the Graphical Device Interface (GDI) drawing functions.

2. The Windows Media Player SDK. Included with the SDK is the documentation you are reading right now, as well as the Windows Media Player Plug-in Wizard.

3. Windows Media Player. You will not be able to test your custom visualization without it. It is also a great way to listen to music while you are coding!

The following sections tell you what you need to know to create a visualization using the plug-in wizard:

- **Getting Started with Visualizations**
- **Using the Visualization Plug-in Wizard with Visual C++ 6.0**
- **Using the Visualization Plug-in Wizard with Visual C++ .NET**

**See Also**

- **About Custom Visualizations**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

### Getting Started with Visualizations

To set up your development environment for creating custom visualizations, you must install the following items:

- Microsoft Visual C++ 6.0, or Microsoft Visual C++ .NET.
- Windows Media Player 9 Series or later.
- Windows Media Player 9 Series SDK or later
- Windows Media Player Plug-in Wizard

#### Installing Visual C++

Be sure you install Microsoft Visual C++ 6.0 or later. Follow the instructions that come with the compiler. Include the Platform SDK for 32-bit Windows because you will need specific information about the Graphics Display Interface (GDI) functions.

#### Installing Windows Media Player

Install Windows Media Player 9 Series or later. It will be needed to test your visualization.

#### Installing the Windows Media Player SDK
Besides reading the programming information, you will need the SDK to get the Windows Media Player Plug-in Wizard. Make sure that the SDK is at the same version level as Windows Media Player, or later.

**Installing the Wizard**

There are separate versions of the Windows Media Player plug-in wizard for Microsoft Visual C++ 6.0 and Microsoft Visual C++ .NET.

Once you have installed the Windows Media Player SDK, you can find the Windows Media Player Plug-in Wizard for Visual C++ 6.0 in this directory:

\wizards\wmpplugin

The wizard file is named wmpwiz.awx.

Copy the wizard to this directory on your hard disk:

\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\IDE

You can also install it in this directory:

\Program Files\Microsoft Visual Studio\Common\MSDev98\Template

If you installed Microsoft Visual Studio 6.0 (which contains Microsoft Visual C++) in another directory, find the corresponding directory. If you are not sure where you installed it, you can search for other COM wizards with the .awx file name extension.

If you are using Microsoft Visual C++ .NET, the Windows Media Player plug-in wizard is installed and configured automatically for you when you install this SDK. If you install Visual C++ .NET after installing this SDK, you should remove the SDK and reinstall it.

**See Also**

- [Building a Visualization](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Using the Visualization Plug-in Wizard with Visual C++ 6.0**

After you have installed the necessary components, creating a sample visualization plug-in is easy. The
following steps will guide you:

2. From the File menu, click New.
4. In the Project name box, type a name for your visualization plug-in.
5. In the Location box, provide the location of a folder where the wizard can save the files it generates. You can accept the default, type a new path, or browse to an existing location.
6. Click OK.
7. Visual C++ displays a dialog box to allow you to choose the type of plug-in you want to create. Visualization should be selected by default, or select it yourself. Click Next.
8. Choose whether your plug-in project should provide code to support a property page.
9. Choose whether your plug-in project should provide code to handle events.
10. Provide a Friendly name and Description for your visualization.
11. Click Finish.
12. Visual C++ displays a dialog box showing you what will be created. You will see that you will be creating a DLL for Win32 and you will be told what kinds of files will be created as source files. You will also see that your project will be placed in the location you specified earlier.
13. Click OK to complete the process.

Do a Test Build

From the Build menu, select Build projectname.dll, where projectname is the project name you chose. Your visualization will be built, and you can watch the progress of the build at the bottom of the screen.

Run Windows Media Player

Start a song in Windows Media Player and click through the visualizations. The presets that are automatically built for you are called projectname Bars and projectname Wave, where projectname is the name of your project.

See Also

- Building a Visualization

Using the Visualization Plug-in Wizard with Visual C++ .NET

Once you have installed the Plug-in Wizard, start up Microsoft Visual C++ .NET. The plug-in wizard can
generate all the code you need to get started. The following steps will guide you:

1. Start up Visual C++.
2. From the File menu, point to New and then click Project.
3. In Project Types, click Visual C++ Projects if it isn't already selected.
4. In Templates, click Windows Media Player Plug-in Wizard to select it.
5. Type a name for your project.
6. Specify a location for your project. This is the folder to which your project files will be copied.
7. Click OK to start the wizard.
8. Visual C++ displays a dialog box to allow you to choose the type of plug-in you want to create. Visualization should be selected by default, or select it yourself. Click Next.
9. Choose whether your plug-in project should provide code to support a property page.
10. Choose whether your plug-in project should provide code to handle events.
11. Provide a Friendly name and Description for your visualization.
12. Click Next.
13. Visual C++ creates a new visualization project for you. The wizard dialog box closes automatically.

Do a Test Build

From the Build menu, select Build projectname, where projectname is the project name you chose. Your visualization will be built, and you can watch the progress of the build at the bottom of the screen.

Run Windows Media Player

Start a song in Windows Media Player and click through the visualizations. The presets that are automatically built for you are called projectname Bars and projectname Wave, where projectname is the name of your project.

See Also

- Building a Visualization

Implementing Your Code

Once you have completed the build process, you will probably want to improve upon the code generated by the wizard. Otherwise your visualization will look much like Bars. What you need to do is change the implementation of the Render function, and possibly the RenderWindowed function.

The following sections provide details about how to modify the sample code generated by the plug-in wizard:
Implementing Render

The easiest way to think of visualization programming is that you are creating a handler for a timed event. At specific intervals, Windows Media Player takes a snapshot of the audio data it is playing, and provides the snapshot data to the currently loaded visualization. This is similar to event-driven programming and is part of the programming model of Microsoft Windows. You write code and wait for it to be called by a particular event.

If your code is an implementation of the `IWMPEffects::Render` function for rendering in windowless mode, it receives the following parameters:

**TimedLevel**

This is a structure that defines the audio data your code will be analyzing. The structure is composed of two arrays. The first array is based on frequency information and contains a snapshot of the audio spectrum divided into 1024 portions. Each cell of the array contains a value from 0 to 255. The first cell starts at 20 Hz and the last at 22050 Hz. The array is two dimensional to represent stereo audio. The second array is based on waveform information and corresponds to audio power, where the stronger the wave is, the larger the value. The waveform array is a granular snapshot of the last 1024 slices of audio power taken at very small time intervals. This array also is two dimensional to represent stereo audio.

**HDC**

This is a Microsoft Windows handle to a device context. This gives a way to identify the drawing surface to Windows. You do not need to create it, you just need to use it for specific drawing function calls.

**RECT**

This is a Microsoft Windows rectangle that defines the size of a drawing surface. This is a simple rectangle with four properties: `left`, `right`, `top`, and `bottom`. The actual values are supplied by Windows Media Player so that you can determine how the user or skin developer has sized the window you will draw on. It also determines the position on the HDC that the effect is supposed to render on.
If your code is an implementation of the `IWMPEffects2::RenderWindowed` function for rendering in a window, it receives the following parameters:

**TimedLevel**

This is the same information that the `Render` function receives.

**fRequiredRender**

The `fRequiredRender` parameter informs you that your visualization must repaint itself—for example, when another window is dragged over it. When this value is false, you can safely skip over the rendering code if the current media item is stopped or paused. This lets you avoid consuming CPU cycles unnecessarily.

The sample plug-in generated by the Plug-in Wizard does not provide a custom implementation for `RenderWindowed`. Instead, the sample plug-in code retrieves the device context from the parent window provided by Windows Media Player in `IWMPEffects2::Create`, then retrieves the dimensions of the parent window as a RECT structure, and then calls through to `Render` using the device context, the RECT, and the timed level pointer from `RenderWindowed`.

The following sections provide more information about implementing `Render`:

- **Using Timed Levels**
- **Using Device Contexts**
- **Using Rectangles**
- **Sample Render Code**

**See Also**

- **Implementing Your Code**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Using Timed Levels**

The `TimedLevel` structure is composed of two two-dimensional arrays, a state value, and a time stamp value.

**Frequency Array**

The frequency array is a two-dimensional array. The first dimension of each array corresponds to the stereo audio channel (left or right), and the second corresponds to the frequency levels (in bytes) of the snapshot, where the audio spectrum is divided up into 1024 regions.
You can get the frequency array data supplied from the Windows Media Player in the following manner:

```c
TimedLevel *pLevels;
int snapshot = pLevels->frequency[0][0];
```

The value of snapshot is for the left channel and contains the value of the lowest part of the frequency spectrum. For example, if snapshot has a large value, it indicates that the lowest 1024th part of the frequency spectrum is rich in frequency. A value of zero indicates no low frequency values in that part of the spectrum for the left channel. If you have a monophonic signal, only the first dimension has valid values.

If the signal is non-stereo, then the second array will contain a copy of the mono signal. That is, frequency[0][n] and frequency[1][n] will contain the same data, where `n` is the index into a particular cell.

**Waveform Array**

The waveform array is also a two-dimensional array. The first dimension of the array corresponds to the channel (left or right), and the second corresponds to the power levels (in bytes) of the snapshot, where the audio power is broken up into 1024 contiguous time segments.

You can get the waveform array data from Windows Media Player in the following manner:

```c
TimedLevel *pLevels;
int snapshot = pLevels->waveform[0][0];
```

The value of snapshot is for the left channel and contains the first value of the quantized snapshot of the power values. When a snapshot is taken, it consists of 1024 tiny incremental measurements of the audio power. The lowest value of the array is generated by the first incremental measurement of audio power. Note that the values of the power are measured from -128 to +127 but the values in the array range from 0 to 255. If you have a monophonic wave, only the first dimension will have valid values.

If the signal is non-stereo, then the second array will contain a copy of the mono signal. That is, waveform[0][n] and waveform[1][n] will contain the same data, where `n` is the index into a particular cell.

**State**

The state variable reflects the audio playback state of Windows Media Player. The PlayerState enumeration values are

```c
stop_state = 0, // audio is currently stopped
pause_state = 1, // audio is currently paused
play_state = 2    // audio is currently playing
```

You can use this variable to take different actions depending on the audio playback state. For example, you can play one kind of visualization when the audio is playing and another when it is stopped.

**Time Stamp**

The timeStamp variable reflects the current time when the snapshot is taken. This can be used to measure how frequently the snapshots are taken.

You can use this variable to time your animations. If the snapshots are too frequent, you can gracefully degrade your image to display in the manner you choose.
Using Device Contexts

The device context is a standard handle to a device context. You need this for many drawing functions so that Microsoft Windows knows which window to draw in. For example, to draw a rectangle, you need to specify the device context.

```c
HDC hdc;
::Rectangle( hdc, 1, 1, 100, 100 );
```

The device context is specified by Windows Media Player through the `Render` function. If your plug-in renders using a window, you'll need to use the device context of that window. Use this device context for any drawing tool that requires a device context.

See Also

- [Implementing Render](#)

Using Rectangles

Rectangles are used to specify rectangular areas in Microsoft Windows. You can create many rectangles in your window, but Windows Media Player supplies the values of one rectangle through the `IWMPEffects::Render` function. If your plug-in renders using a window, the rectangle is the client area of the window. This is called the prc rectangle, and it defines the rectangle that Windows Media Player will display your visualization through. Use this frequently to be sure you do not draw beyond the extents of the rectangle supplied by Windows Media Player.
A rectangle has four values that define it. They are left, top, right, and bottom. The top, left corner of the rectangle is defined by left and top, and the bottom, right corner of the rectangle is defined by bottom and right.

Use the following code to get the extents of your drawing rectangle. You need to do this because the user may resize the window, and you want to be sure that you are always drawing in an area the user can see.

```c
int leftside = prc->left;
int rightside = prc->right;
int topside = prc->top;
int bottomside = prc->bottom;
```

For example, to draw from left to right, along the top of the window, use code like this:

```c
::MoveToEx( hdc, prc->left, prc->top, NULL );
::LineTo(hdc, prc->right, prc->top);
```

See Also

- [Implementing Render](#)

### Sample Render Code

Here is some sample code that uses the `Render` function to draw a line across the screen. The height of the line is defined by the waveform value.

```c
STDMETHODIMP CStock::Render(TimedLevel *pLevels, HDC hdc, RECT *prc)
{
    // Create new brushes and pens.
    HBRUSH hNewBrush = ::CreateSolidBrush( 0 );
    // Create a new solid pen the color of the foreground.
    HPEN hNewPen = ::CreatePen( PS_SOLID, 0, m_clrForeground );

    // Add the pen to the device context.
    HPEN holdPen= static_cast<HPEN>(::SelectObject( hdc, hNewPen ));

    // Fill the background with the black brush.
    ::FillRect( hdc, prc, hNewBrush );

    // Get the y value from the waveform.
    int y = pLevels->waveform[0][0];

    // Draw the line from left to right.
    ::MoveToEx( hdc, prc->left, y, NULL );
    ::LineTo(hdc, prc->right, y);
}
```
// Delete your brush.
if (hNewBrush)
{
    ::DeleteObject( hNewBrush );
}
// Delete your pen.
if (hNewPen)
{
    ::SelectObject( hdc, hOldPen );
    ::DeleteObject( hNewPen );
}

// You're done for this round.
return S_OK;

The **Render** function is where the main work of your code takes place. Every time Windows Media Player takes a snapshot of the audio, it will call this function and your code will run.

This code performs the following tasks. Refer to the Microsoft Windows Platform SDK for 32-bit Windows for more details about specific functions.

**Creating Objects**

Usually you will be using the drawing functions that come with the Microsoft Windows Graphical Display Interface (GDI). You need to create pens to draw lines and brushes to fill areas.

A solid black brush is created to fill in the background.

A solid pen is created to draw a line. The color will be the foreground color as defined by the skin that is going to display the visualization.

**Adding the Object to the DC**

You need to add the pen to the device context (DC). The DC is the portion of memory that all drawing data and objects are stored in. Essentially the DC is the window traffic manager that keeps track of everything graphical.

You need to *cast* the pen object you created and store it as an old pen. Use this coding technique for all new pens. This technique is required for 32-bit programming.

**Filling in the Background**

You now are ready to draw. The **FillRect** function will fill the rectangle of the window, as defined by the parameters of the **Render** function. The rectangle is filled with a black brush.

**Getting Audio Data**

Next the code gets some audio data from Windows Media Player. By using the waveform array, you can get the current value of the audio power at the moment the snapshot was taken. In this case, you are taking the audio data of the left channel. The first value in the array is the first 1024th of the audio power snapshot.

This information will be used to display a line whose height will match the audio power snapshot.

**Draw the Line**
The line is drawn from left to right using the **MoveToEx** and **LineTo** GDI functions.

First you move the pen to the starting point. In this case, x and y are used to define the left-to-right and top-to-bottom values the user will see on the screen. X is defined by the rectangle prc and specifically by the value of prc->left. Y is defined as the value of the waveform data at that moment.

Then you draw a line to the other side of the window. The point you draw the line to is again an x, y value. X is defined by the rectangle prc, but this time by prc->right. Y is still defined by the waveform data and is the same as the point you started from, because you are drawing a straight line from left to right.

**Clean up Everything**

You must delete the objects you create. Specifically you must delete any and all brushes and pens you create. It is good practice to delete pens and brushes as soon as you finish using them.

If you do not delete them before finishing your implementation of the **Render** function, your visualization will crash in a minute or less. You must keep a count of your pens and brushes and destroy every single one. Be especially careful not to create pens inside a code loop.

Use the coding technique given in the example to destroy your pens and brushes.

**Important**  Destroy your pens and brushes!

When you have finished cleaning up, be sure to return S_OK so that Windows Media Player knows you are finished drawing. Once you finish, your drawing will be transferred to the window, another snapshot will be taken, **Render** will ask your code to draw again, and so on.

**See Also**

- [Implementing Render](#)

---

**Implementing Other Functions**

You will definitely want to write your own implementation of the **Render** function. Several other functions that are also member functions of the **IWMPEffects** interface are provided. Some will provide you with extra information that you can choose to use and others will automatically provide Windows Media Player with information that was generated by the wizard, such as the name of your visualization.

The **IWMPEffects** interface supports the following functions in addition to **Render**:
The **IWMPEffects2** interface supports the following additional functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DisplayPropertyPage</strong></td>
<td>Default implementation not supplied by wizard.</td>
</tr>
<tr>
<td><strong>GetCapabilities</strong></td>
<td>Gets the capabilities of your visualization and passes them to Windows Media Player.</td>
</tr>
<tr>
<td><strong>GetCurrentPreset</strong></td>
<td>The wizard created two presets when it generated the code for your visualization. This function is called when the skin developer wants to get the index of the current preset. You will not want to change the implementation of this function because it just uses information set by other functions.</td>
</tr>
<tr>
<td><strong>GetPresetCount</strong></td>
<td>The wizard created two presets when it generated the code for your visualization. You can change the count by changing the implementation of <strong>GetPresetCount</strong>. See <a href="#">Presets</a> for more information about changing the presets.</td>
</tr>
<tr>
<td><strong>GetPresetTitle</strong></td>
<td>The wizard created two presets when it generated the code for your visualization. You can change the titles used by changing the implementation of <strong>GetPresetTitle</strong>. See <a href="#">Presets</a> for more information about changing the presets.</td>
</tr>
<tr>
<td><strong>GetTitle</strong></td>
<td>Gets the title of your visualization and passes it to Windows Media Player. The wizard used the name of your project to generate the name that is passed back.</td>
</tr>
<tr>
<td><strong>GoFullscreen</strong></td>
<td>Default implementation not supplied by wizard.</td>
</tr>
<tr>
<td><strong>MediaInfo</strong></td>
<td>Retrieves the number of audio channels and the sample rate of the audio currently playing.</td>
</tr>
<tr>
<td><strong>RenderFullScreen</strong></td>
<td>Default implementation not supplied by wizard.</td>
</tr>
<tr>
<td><strong>SetCurrentPreset</strong></td>
<td>The wizard created two presets when it generated the code for your visualization. This function is called when the Windows Media Player wants to change to a named preset.</td>
</tr>
</tbody>
</table>

The **IWMPEffects2** interface supports the following additional functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>When rendering in a window, Windows Media Player calls this function to allow you to create a new window for rendering.</td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td>When rendering in a window, Windows Media Player calls this function to permit you to destroy the window you created when <strong>Create</strong> was called.</td>
</tr>
<tr>
<td><strong>NotifyNewMedia</strong></td>
<td>This function allows your visualization to respond when a new media item has been loaded by the Player.</td>
</tr>
<tr>
<td><strong>OnWindowMessage</strong></td>
<td>This function receives windows messages from the Player when rendering in windowless mode.</td>
</tr>
<tr>
<td><strong>RenderWindowed</strong></td>
<td>This function is called by the Player instead of <strong>IWMPEffects::Render</strong> when the Player is rendering in windowed mode.</td>
</tr>
</tbody>
</table>
Presets

Presets are provided as a way to have different effects coming from the same visualization. For example, you could create a glow effect that was generated by one block of code, but use a preset to determine the color of the glow. Your preset names might be Red, Green, and Blue.

The wizard defines two presets for the code it generates. One is called Bars and the other is called Wave. The Bars preset displays bars that show the activity in the audio spectrum, and use the waveform data. The Wave preset displays a wiggling line that shows the audio power of the waveform.

If you change any of the preset information, you must also change the following parts of the generated code.

Render function

The \texttt{IWMPEffects::Render} function is in the file \texttt{projectname.cpp}, where \texttt{projectname} is the project name you chose when you ran the wizard.

The generated code in the \texttt{Render} function uses a switch statement to choose between two presets. The current preset is whatever the user selects in Windows Media Player. If you want to change which code runs for a particular preset or add or subtract a preset, modify the switch statement accordingly.

The two presets are defined by the \texttt{PRESET_BARS} and \texttt{PRESET_SCOPE} enumerations. The choice of which preset will be called is defined by \texttt{m_nPreset}.

GetPresetTitle

The \texttt{IWMPEffects::GetPresetTitle} function is in the file \texttt{projectname.cpp}, where \texttt{projectname} is the project name you chose when you ran the wizard.

The \texttt{GetPresetTitle} function sets up the relationships between the preset enumerations and the string resources. The enumerations \texttt{PRESET_BARS} and \texttt{PRESET_SCOPE} are generated by the wizard and are using the string resources \texttt{IDS_BARSPRESETNAME} and \texttt{IDS_SCOPEPRESETNAME}. 
You need to change the enumerations and string resources if you add, subtract, or change presets.

**Preset Enumerations**

The preset enumeration is defined in the file *projectname*.h, where *projectname* is the project name you chose when you ran the wizard.

The enumeration defines the current two presets and the count. If you add or subtract presets or change the enumeration, be sure you change this enumeration so that the count and order of presets is correct. This enumeration is used to make sure that you call the correct preset in the `Render` function.

**Resource Header**

You must set the resources for names of your preset in the `resource.h` header file. The current presets are defined as:

```c
#define IDS_BARSPRESETNAME 102
#define IDS_SCOPEPRESETNAME 103
```

If you add or subtract presets, you must change the resource header and the numbers for them.

**Resource Strings**

The actual names of the presets are defined in the resource file *projectnamedll.rc*, where *projectname* is the project name you chose when you ran the wizard. You can edit this file by hand or use the resource editor included with Microsoft Visual C++.

The names generated are the name of the visualization plus the specific preset. The resource file for the generated code will define them as:

```c
IDS_BARSPRESETNAME      "projectname Bars"
IDS_SCOPEPRESETNAME     "projectname Wave"
```

where *projectname* is the name of the project name you chose when you ran the wizard. This is where you will change the actual names of the presets and this is how they will be referred to and displayed by Windows Media Player.

See Also

- Implementing Your Code
Installing Visualizations

You must provide an installation process for the user of your visualization. You must also provide an uninstall process for the user. The current version of Windows Media Player does not install visualizations from the user interface.

Installing to the Visualization Folder

It is recommended that you install all visualizations in the Visualizations subfolder of the folder where Windows Media Player is installed.

Registering Your Visualization

Visualizations are COM DLLs and follow all the normal rules of installation and deinstallation. You can use regsvr32.exe or other installation tools to register your visualization.

See Also

- About Custom Visualizations

Submitting Your Visualization

If you want to submit your visualization to the Windows Media Web site, do this:

Test Your Visualization

You may want to try installing your visualization on several machines to be sure that everything works the way you want. Try every preset to make sure your code works correctly.

Sign Your Visualization

You must submit your visualization for code signing so that the user can decide whether or not to trust your visualization before they install it. You can get full details about code signing here.

Distributing Your Visualization

Consult the Windows Media Web Gallery for more information on how to distribute your visualization.
Writing your own visualization is as simple as implementing the `Render` function, modifying preset information, and building your DLL.

The following topic describes how to create a sample visualization.

**Topic**  
The Glow Sample

**Description**  
Describes how to create a simple visualization.

The Glow Sample  
This section describes how to create a simple visualization called Glow. Three presets are used: Red, Green, and Blue. The Custom visualization wizard was run and the name Glow was chosen.

Glow simply changes color in time to the music. The glow flickers because the audio power of the waveform sometimes crosses zero at the moment of the snapshot. This sample is chosen because it is one of the simplest...
Implementing Render

The following code is used to implement the `Render` function:

```cpp
STDMETHODIMP CGlow::Render(TimedLevel *pLevels, HDC hdc, RECT *prc)
{
    COLORREF mycolor;
    int mylevel = pLevels->waveform[0][0];

    switch (m_nPreset)
    {
    case PRESET_RED:
        {
            mycolor = RGB( mylevel, 0, 0);
        } break;
    case PRESET_GREEN:
        {
            mycolor = RGB( 0, mylevel, 0);
        } break;
    case PRESET_BLUE:
        {
            mycolor = RGB( 0, 0, mylevel);
        } break;
    }

    HBRUSH hNewBrush = ::CreateSolidBrush( mycolor );
    ::FillRect( hdc, prc, hNewBrush );

    if (hNewBrush)
    {
```
Here is an explanation of the code:

A variable named `mycolor` is used for the color of the glow and is declared with `COLORREF`. All colors should use the `COLORREF` data type.

A variable named `mylevel` is used for the audio waveform level snapshot. This value will depend on the actual power level at the moment of the snapshot.

The `switch` statement is set by the preset that the user has chosen on Windows Media Player. The choice will set `mycolor` to the desired color (red, green, or blue). However, the exact color will be determined by the audio power level. For example, if the red preset is chosen, the color will be a solid red, but it will be lighter or darker depending on the audio waveform at the moment of the snapshot. Be sure to use the `RGB` macro to create your color.

A brush is created called `hNewBrush`, and it is used to fill the `prc` rectangle provided by Windows Media Player. The drawing surface is the `hdc` device context provided by Windows Media Player.

The brush is deleted by `DeleteObject`. Always be sure to delete any pens or brushes you create.

Once the `Render` code is finished, Windows Media Player will display the `hdc` graphics in a window determined by the skin being used.

See Also

- The Glow Sample

© 2000-2003 Microsoft Corporation. All rights reserved.

Changing Presets

The following preset code sections were changed to allow three presets:

GetPresetTitle

This code was inserted in place of the generated preset code:
switch (nPreset)
{
    case PRESET_RED:
        bstrTemp.LoadString(IDS_REDPRESETNAME);
        break;

    case PRESET_GREEN:
        bstrTemp.LoadString(IDS_GREENPRESETNAME);
        break;

    case PRESET_BLUE:
        bstrTemp.LoadString(IDS_BLUEPRESETNAME);
        break;
}

Enumerations

The following enumeration in Glow.h was changed to allow three presets:

    enum {
        PRESET_RED = 0,
        PRESET_GREEN,
        PRESET_BLUE,
        PRESET_COUNT
    };

Resource Header

The following resources were defined in Resource.h to allow three presets:

    #define IDS_REDPRESETNAME               102
    #define IDS_GREENPRESETNAME             103
    #define IDS_BLUEPRESETNAME              104

Note that you must also change the resource number of _APS_NEXT_SYMED_VALUE to 106.

Resource File

The following strings must be changed in the Glowdll.rc file to allow three presets and give them names:

    IDS_REDPRESETNAME       "Glow Red"
    IDS_GREENPRESETNAME     "Glow Green"
    IDS_BLUEPRESETNAME      "Glow Blue"

See Also

    • The Glow Sample
Custom Visualization Programming Reference

The Custom Visualization Reference documents the following items.

IWMPEffects Interface
An interface to custom visualizations.

IWMPEffects2 Interface
An interface that extends IWMPEffects, allowing greater control over visualization behavior.

Visualization Structures and Enumeration Types
Data types for use in manipulating custom visualizations.

IWMPEffects Interface

In addition to the methods inherited from IUnknown, the IWMPEffects interface exposes the following methods, in IDL order.

Method | Description
--- | ---
Render | Renders the visualization
MediaInfo | Sends channel and sample-rate data to the visualization.
GetCapabilities | Gets the capabilities of the visualization.
GetTitle | Gets the display title of the visualization.
GetPresetTitle | Gets the title of the current preset.
GetPresetCount | Gets the preset count.
SetCurrentPreset | Sets the current preset by number.
The `DisplayPropertyPage` method displays the property page of a visualization, if it exists.

**Syntax**

```cpp
HRESULT DisplayPropertyPage(
    [in] HWND hwndOwner
);```

**Parameters**

`hwndOwner`

[in] Handle to the dialog that will be displayed.

**Return Values**

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

**Remarks**

Implement this method if you want to display a property page to the user to adjust any values of the visualization.

**Requirements**
IWMPEffects::GetCapabilities

The GetCapabilities method gets the capabilities of the visualization.

Syntax

HRESULT GetCapabilities(
    DWORD* pdwCapabilities
);

Parameters

pwdCapabilities

[out] DWORD containing the capabilities.

The current values are as follows.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFFECT_CANGOFULLSCREEN = 0x00000001;</td>
<td>The visualization is capable of full-screen rendering.</td>
</tr>
<tr>
<td>EFFECT_HASPROPERTYPAGE = 0x00000002;</td>
<td>The visualization has a property page.</td>
</tr>
<tr>
<td>EFFECT_VARIABLEFREQSTEP = 0x00000004;</td>
<td>The visualization will use frequency data with variable size steps. If this bit is set, step size is based on the media sampling frequency divided by BUFFER_SIZE. If this bit is not set and media is played that was sampled at a low frequency, the upper cells will be empty. For example, if an 8KHz...</td>
</tr>
</tbody>
</table>
The **GetCurrentPreset** method gets the current preset, by number, from the visualization and provides it to Windows Media Player.

sampled file is played and this bit is not set, the upper half of the frequency array (from 8KHz to 22KHz) will be empty. If this bit is set and an 8Khz sampled file is played, the frequency array will range from 20Hz to 8KHz in BUFFER_SIZE steps.

**EFFECT_WINDOWED_ONLY** = 0x00000008

The visualization only renders in windowed mode.

**EFFECT2_FULLSCREENEXCLUSIVE** = 0x00000010

The visualization uses exclusive mode when rendering full-screen. The Player will not resize the window to fill the screen. The visualization must create a top level window and handle resolution switching.

### Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an **HRESULT** error code.

### Remarks

A default implementation of this method is not included in the visualization wizard.

### Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in effects.idl; include effects.h.

**See Also**

- **IWMPEffects Interface**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPEffects::GetCurrentPreset**

The **GetCurrentPreset** method gets the current preset, by number, from the visualization and provides it to Windows Media Player.
Syntax

HRESULT GetCurrentPreset(
    Long* currentpreset
);

Parameters

currentpreset

[in] Long value specifying the current preset, by number.

Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPEffects::GetPresetCount

The GetPresetCount method gets the preset count.

Syntax

HRESULT GetPresetCount(
    Long* count
);

Parameters

count
Long value specifying the preset count.

Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Remarks

Called by Windows Media Player to obtain the number of presets contained by the visualization.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

IWMPEffects::GetPresetTitle

The GetPresetTitle method gets the title of the current preset.

Syntax

```cpp
HRESULT GetPresetTitle(
    Long nPreset,
    BSTR* bstrPresetTitle
);
```

Parameters

- **nPreset**
  
  [in] Long preset number.

  **bstrPresetTitle**

  [out] BSTR preset title.
Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Remarks

Called by Windows Media Player to obtain the title of a given preset.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface

IWMPEffects::GetTitle

The GetTitle method gets the display title of the visualization.

Syntax

HRESULT GetTitle(
    BSTR* bstrTitle
);

Parameters

bstrTitle

[out] String containing the title.

Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Remarks
This method allows the application to show the user a title when the visualization itself is displayed. The title should be as unique as possible. Do not use titles of visualizations included with the Windows Media Player as this will cause confusion to the user.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

• IWMPEffects Interface

IWMPEffects::GoFullscreen

The GoFullscreen method instructs the visualization to switch to full-screen mode.

Syntax

HRESULT GoFullscreen(
    BOOL  fFullscreen
);

Parameters

fFullscreen

[in] Boolean indicating whether to switch to full-screen mode.

Return Values

If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Remarks

This method is called when the visualization is to go full screen or leave full screen. If the full screen capabilities flag is not returned through GetCapabilities, your visualization will not be asked to go or render full screen. This method will be called before RenderFullScreen is called.
A default implementation of this method is not included in the visualization wizard.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface

IWMPEffects::MediaInfo

The MediaInfo method sends channel and sample rate data to the visualization.

Syntax

HRESULT MediaInfo(
    LONG lChannelCount,
    LONG lSampleRate,
    BSTR bstrTitle
);

Parameters

lChannelCount

[in] Long integer containing the number of channels (one for mono, or two for stereo).

lSampleRate

[in] Long integer containing the sample rate in hertz (Hz). For example, a value of 22500 would specify a rate of 22.5KHz.

bstrTitle

[in] String specifying the title.

Return Values
If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- **IWMPEffects Interface**

IWMPEffects::Render

The `Render` method renders the visualization.

Syntax

```cpp
HRESULT Render(
    TimedLevel* pLevels,
    HDC hdc,
    RECT* p rect
);
```

Parameters

- `pLevels`
  
  [in] Pointer to a `TimedLevel` structure.

- `hdc`
  
  [in] Specifies a handle to a device context.

- `prc`
  
  [in] Specifies the rectangle the visualization is to be rendered in.

Return Values
If the method succeeds, it returns S_OK. If it fails, it returns an HRESULT error code.

Remarks

The device context is normalized by this method.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface
- TimedLevel

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPEffects::RenderFullScreen

The RenderFullScreen method renders the visualization in full-screen mode.

Syntax

HRESULT RenderFullScreen(
    TimedLevel* pLevels,
);

Parameters

pLevels

[in] Pointer to a TimedLevel structure.

Return Values

If the method succeeds, your implementation should return S_OK. If it fails, return an HRESULT error code.

Remarks

The GoFullscreen method must be called with a True value before RenderFullScreen can be called.
The user can enter or leave full screen mode by pressing the Alt and Enter keys simultaneously.

A default implementation of this method is not included in the visualization wizard.

If your implementation returns an error from this method, then `GoFullscreen(False)` will be called to ask your visualization to drop out of full screen mode.

Requirements

**Version**: Windows Media Player version 7.0 or later.

**Header**: Defined in effects.idl; include effects.h.

See Also

- `IWMPEffects Interface`
- `TimedLevel`

IWMPEffects::SetCurrentPreset

The `SetCurrentPreset` method gets the current preset from Windows Media Player and sets it in the visualization.

**Syntax**

```c
HRESULT SetCurrentPreset(
    Long currentpreset
);
```

**Parameters**

- `currentpreset`[out]  `Long` value specifying the new preset index.

**Return Values**

If the method succeeds, it returns S_OK. If it fails, it returns an `HRESULT` error code.

**Remarks**
This is called by the Windows Media Player to request that the given preset be displayed.

Requirements

Version: Windows Media Player version 7.0 or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects Interface

IWMPEffects2 Interface

In addition to the methods inherited from IWMPEffects, the IWMPEffects2 interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Called by Windows Media Player to instantiate a visualization window.</td>
</tr>
<tr>
<td>Destroy</td>
<td>Called by Windows Media Player to destroy a visualization window</td>
</tr>
<tr>
<td>NotifyNewMedia</td>
<td>Called by Windows Media Player to inform the visualization that a new</td>
</tr>
<tr>
<td>OnWindowMessage</td>
<td>Called by Windows Media Player to pass window messages to a visualization.</td>
</tr>
<tr>
<td>RenderWindowed</td>
<td>Called by Windows Media Player to render a windowed visualization.</td>
</tr>
<tr>
<td>SetCore</td>
<td>Called by Windows Media Player to provide visualization access to the</td>
</tr>
<tr>
<td></td>
<td>core Windows Media Player APIs.</td>
</tr>
</tbody>
</table>

See Also

- Custom Visualization Programming Reference
- IWMPEffects Interface

© 2000-2003 Microsoft Corporation. All rights reserved.
The `Create` method is called by Windows Media Player to instantiate a visualization window.

**Syntax**

```c
HRESULT Create(
    HWND hwndParent
);
```

**Parameters**

- `hwndParent`  
  [in] HWND handle to the parent window hosting the visualization window.

**Return Values**

This method returns an `HRESULT`.

**Remarks**

A visualization that implements `IWMPEffects2` is rendered in its own window unless it will be displayed in a clipped device context, in which case it is rendered windowless. For a visualization that is rendered windowless, Windows Media Player calls this method with a NULL value for the `hwndParent` parameter. If your visualization does not support windowless mode (for example, when using Direct 3D), it should return a failure `HRESULT` value. In this case, your visualization will not be available in skins that clip the display region.

If you create a visualization for Windows Media Player using the Direct3D® component of Microsoft DirectX®, you must set the `D3DCREATE_FPU_PRESERVE` flag when calling `IDirect3D8::CreateDevice`. Failure to set this flag for visualizations that use Direct3D may yield unexpected results.

**Requirements**

- **Version:** Windows Media Player 9 Series or later.
- **Header:** Defined in effects.idl; include effects.h.

**See Also**

- `IWMPEffects2` Interface
The **Destroy** method is called by Windows Media Player to destroy a visualization window instantiated in the **Create** method.

**Syntax**

```c
HRESULT Destroy();
```

**Parameters**

This method takes no parameters.

**Return Values**

This method returns an **HRESULT**.

**Remarks**

This method is used only by windowed visualizations. Windowless visualizations should simply return S_OK.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in effects.idl; include effects.h.

**See Also**

- IWMPEffects2 Interface
- IWMPEffects2::Create
The NotifyNewMedia method is called by Windows Media Player to inform the visualization that a new media item has been loaded.

**Syntax**

```c
HRESULT NotifyNewMedia(IWMPMedia* pMedia);
```

**Parameters**

*pMedia*

[in] Pointer to an IWMPMedia interface that represents the new media item.

**Return Values**

This method returns an HRESULT.

**Requirements**

- **Version:** Windows Media Player 9 Series or later.
- **Header:** Defined in effects.idl; include effects.h.

**See Also**

- [IWMPEffects2 Interface](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.
The **OnWindowMessage** method is called by Windows Media Player to pass window messages to a visualization.

## Syntax

```
HRESULT OnWindowMessage(  
    UINT msg,  
    WPARAM WParam,  
    LPARAM LParam,  
    LRESULT* plResultParam  
);
```

## Parameters

- **msg**
  
  [in] **UINT** that identifies the window message.

- **WParam**

  [in] **WPARAM** specifying a window message parameter.

- **LParam**

  [in] **LPARAM** specifying a window message parameter.

- **plResultParam**

  [in] Pointer to an **LRESULT** specifying the result code for the window message.

## Return Values

This method returns an **HRESULT**.

## Remarks

Your implementation must only return S_OK if it has handled the window message. If it has not handled the window message, it should return S_FALSE. If this method is not implemented, return E_NOTIMPL.

## Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in effects.idl; include effects.h.

**See Also**

- [IWMPEffects2 Interface](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
IWMPEffects2::RenderWindowed

The `RenderWindowed` method is called by Windows Media Player to render a windowed visualization.

Syntax

```c
HRESULT RenderWindowed(
    TimedLevel* pData,
    BOOL  fRequiredRender
);
```

Parameters

- `pData`
  [in] Pointer to a `TimedLevel` structure specifying rendering information.

- `fRequiredRender`
  [in] `BOOL` indicating whether the visualization must paint itself.

Return Values

This method returns an `HRESULT`.

Remarks

This method is used to render windowed visualizations. Windowless visualizations should return S_OK and use the `IWMPEffects::Render` method instead.

The `fRequiredRender` parameter informs you that your visualization must repaint itself, for example, when another window is dragged over it. When this value is false, you can safely skip over the rendering code if the current media item is stopped or paused. This lets you avoid consuming CPU cycles unnecessarily.

Requirements

- **Version**: Windows Media Player 9 Series or later.
- **Header**: Defined in effects.idl; include effects.h.

See Also

- `IWMPEffects::Render`
- `IWMPEffects2` Interface
- `TimedLevel`
IWMPEffects2::SetCore

The SetCore method is called by Windows Media Player to provide visualization access to the core Windows Media Player APIs.

Syntax

HRESULT SetCore(IWMPCore* pPlayer);

Parameters

pPlayer

[in] Pointer to an IWMPCore interface.

Return Values

This method returns an HRESULT.

Remarks

You can use this method to set or release a pointer to the IWMPCore interface. If pPlayer is NULL, the visualization is being shut down and all stored references to the core should be released.

This method is not called when Windows Media Player instantiates the visualization for the purpose of displaying its property page. This method can therefore be used as an entry point that will only be called when the visualization is enabled and Windows Media Player loads it normally.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in effects.idl; include effects.h.

See Also

- IWMPEffects2 Interface
Visualization Structures and Enumeration Types

Structure or Enumeration | Description
--- | ---
**PlayerState** | Provides some basic states of Windows Media Player.
**TimedLevel** | Holds data returned from the spectrum filter.

**PlayerState**

The **PlayerState** enumeration type provides some basic states of Windows Media Player.

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stop state</td>
</tr>
<tr>
<td>1</td>
<td>Pause state</td>
</tr>
<tr>
<td>2</td>
<td>Play state</td>
</tr>
</tbody>
</table>

**Syntax**

```c
typedef enum PlayerState{
    stop_state  = 0,
    pause_state = 1,
    play_state  = 2
};
```
Remarks

This enumeration is used by the **TimedLevel** structure.

Requirements

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in effects.idl; include effects.h.

See Also

- Visualization Structures and Enumeration Types
- TimedLevel

---

TimedLevel

The **TimedLevel** structure holds data returned from the spectrum filter.

Syntax

```c
typedef struct tagTimedLevel{
    unsigned char frequency[2][SA_BUFFER_SIZE];
    unsigned char waveform[2][SA_BUFFER_SIZE];
    int state;
    hyper timestamp;
} TimedLevel;
```

Members

**frequency**

Provides a stereo snapshot of the frequency spectrum of the audio data at a time specified by the Plug-in Manager. It can be used for frequency spectrum effects such as real-time analyzers. The frequency value of the first cell is 20 Hz, and the frequency value of the last cell is 22050 Hz.

**waveform**

Provides a stereo snapshot of the power value of the audio data at a time specified by the Plug-in Manager as the first element; the next 1024 stereo power values fill out the rest of the array. It can be used for oscilloscope-
type effects.

**state**

Specifies one member of the **PlayerState** enumeration type.

**timeStamp**

Specifies the time the snapshot took place, in a 64-bit integer. The time value is provided in 100-nanosecond units.

**Remarks**

The array dimension **SA_BUFFER_SIZE** is currently 1024.

The first dimension of each array corresponds to the channel: 0 is a monaural signal or the left channel of a stereo signal, and 1 is the right channel of a stereo signal. If the signal is monaural, the values in the array that would correspond to the right channel are undefined.

The second dimension contains the sampled levels. The frequency data ranges from 0 to 255. The waveform data represents -128 to 127 but is stored as 0 to 255, so subtract 128 to get the correct value.

This structure is defined in the effects.idl file.

**Requirements**

**Version:** Windows Media Player version 7.0 or later.

**Header:** Defined in effects.idl; include effects.h.

**See Also**

- [PlayerState](#)
- [Visualization Structures and Enumeration Types](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Windows Media Player User Interface Plug-ins**

Microsoft Windows Media Player provides an architecture that enables the user to install and activate plug-in programs that add user interface (UI) functionality to the full mode of the player. A typical UI plug-in might
allow a user to buy the CD that contains the currently playing track or to access additional information about the artist. This section of the Windows Media Player Software Development Kit (SDK) provides you with the programming information you need to create your own UI plug-in.

The UI plug-in documentation is divided into three sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About User Interface Plug-ins</strong></td>
<td>Provides an overview of the architecture used for UI plug-ins. Read this section to learn the general concepts involved with this technology.</td>
</tr>
<tr>
<td><strong>User Interface Plug-ins Programming Guide</strong></td>
<td>Explains what you need to do to create a UI plug-in. This section contains example code and step-by-step procedures.</td>
</tr>
<tr>
<td><strong>User Interface Plug-ins Programming Reference</strong></td>
<td>Provides a detailed reference for the COM interface and methods supported by the Windows Media Player SDK for UI plug-ins.</td>
</tr>
</tbody>
</table>

See Also

- [Windows Media Player Plug-ins](#)

---

About User Interface Plug-ins

Windows Media Player provides a variety of control panels that allow the user to modify various aspects of the player such as the video and graphic equalizer settings. These control panels can enhance the multimedia experience provided by Windows Media Player, but they do not necessarily provide all the functionality that the user may be looking for.

Skins are one way to provide additional functionality, but they require the developer to recreate the entire user interface (UI). As an alternative, Windows Media Player allows the creation of custom UI plug-ins that display in the full mode of the player. This functionality is provided through a programming interface that follows standard Microsoft Component Object Model (COM) guidelines. You can implement this interface in Microsoft Visual C++ by using the Windows Media Player Plug-in Wizard to help you get started.

UI plug-ins are described in greater detail in the following topics.
UI Plug-in Overview

UI plug-ins can be installed, uninstalled, and configured using the Plug-ins tab of the Options dialog in Windows Media Player.

There are five types of UI plug-ins supported by Windows Media Player. Three of these types correspond to different areas of the Now Playing pane of the full mode of the player. The other two types are for plug-ins that display in a separate window and for plug-ins that have no display (except perhaps a property page).

UI plug-ins are unloaded when they are closed or disabled. UI plug-ins that appear in the Now Playing pane are also unloaded when the user chooses another plug-in of the same type or switches out of the Now Playing pane. When the user switches to skin mode, all plug-ins remain loaded, but are not displayed. If separate window or background plug-ins are running when Windows Media Player is closed, they will be automatically reloaded the next time the player is started.

The following sections provide general information about Windows Media Player UI plug-ins:

- Display Area Plug-ins
- Settings Area Plug-ins
- Metadata Area Plug-ins
- Separate Window Plug-ins
- Background Plug-ins
- UI Plug-in Options
- Displaying Modal User Interfaces

See Also

- About User Interface Plug-ins
Display Area Plug-ins

Display area UI plug-ins are shown in the main area of the Now Playing pane, below the artist and track information and to the left of the playlist area. Only one display area plug-in can be enabled at a time. When a display area plug-in is enabled, the Player cannot enter full screen mode.

Display area plug-ins are useful for displaying large amounts of information. For example, a display area plug-in can retrieve information about the currently playing media item and display related articles, interviews, and reviews. A display area plug-in can host a Microsoft Internet Explorer browser window to display this information using HTML.

**Note** A hosted browser window can embed the ActiveX version of Windows Media Player, but this is not recommended because the embedded control cannot communicate with the standalone player that contains it.

**Note** Windows Media Player closes any open display area plug-in when the user plays content that contains video.

See Also

- [UI Plug-in Overview](#)

Settings Area Plug-ins

Settings area UI plug-ins are shown in the Tools area of the Now Playing pane below the main display area and above the transport controls. Only one settings area plug-in can be enabled at a time.

Settings area plug-ins are useful for displaying custom control panels similar to the graphic equalizer or video
settings controls.

See Also

- UI Plug-in Overview

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

 Metadata Area Plug-ins

Metadata area UI plug-ins are shown in the metadata area of the Now Playing pane to the right of the main visual area and the Tools pane, and below the playlist view (if it is enabled). Only one metadata area plug-in can be enabled at a time.

Metadata area plug-ins are useful for displaying concise information about the currently playing media item, simple controls, or hyperlinks to additional information.

See Also

- UI Plug-in Overview

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

 Separate Window Plug-ins

Separate window UI plug-ins are shown in their own windows. These windows can be fixed in size or can be made resizable. Multiple separate UI plug-in windows can be open at the same time.

Separate window UI plug-ins are useful when you want the information or controls you provide to remain visible when the user switches out of the Now Playing pane or loads another UI plug-in. Any separate window UI plug-ins that are running when Windows Media Player is closed will be reloaded and displayed in the same
size and position when the player is started again.

See Also

- UI Plug-in Overview

Background Plug-ins

Background UI plug-ins do not actually have user interfaces unless they implement property pages. When one or more background plug-in is running, a special icon appears in the Windows Media Player display next to the running track time. This icon can be clicked to display the background plug-ins options in the Options dialog box. From there, the user can choose to open the property page of any plug-in that supports one.

Background plug-ins are useful for providing automatic services that do not require user input.

See Also

- UI Plug-in Overview

UI Plug-in Options

Any of the five UI plug-in types can have a property page where the user can modify plug-in settings. This property page can be set to display automatically when a plug-in is loaded for the first time. It can also be accessed from the Plug-ins tab of the Options dialog box.

A UI plug-in can be set to load automatically after installation when Windows Media Player is started. If it is not started automatically, the user can load it by selecting it from the Plug-ins list on the Tools menu.
A display area plug-in can have multiple presets, which are different views that the plug-in can make available to the user. The user can change the preset by selecting a different entry from the Visualizations and Views list on the View menu, or by using the arrow buttons provided at the bottom of the Now Playing pane just above the status message and transport controls.

Background and separate window UI plug-ins can be programmed to accept a media item or playlist that the user sends to it. If a plug-in supports this feature, its name will appear on the Send to list available from the shortcut menu of the playlist control or Media Library.

A UI plug-in can be programmed to intercept keyboard shortcuts when it has the keyboard focus. Keyboard focus is required to prevent conflicts when multiple UI plug-ins that intercept the same keyboard shortcut are loaded. Although this prevents conflicts, it can also cause confusion to end users. For this reason, the use of keyboard shortcuts with UI plug-ins is not recommended. When they are used, however, they should not intercept any keyboard shortcuts that are used by Windows Media Player. For a complete list of keyboard shortcuts used by the Player, see Windows Media Player Help.

See Also

- UI Plug-in Overview

Displaying Modal User Interfaces

UI plug-ins should not display modal dialog boxes or message boxes in response to method calls from Windows Media Player. For example, do not display a message box from your implementation of IWMPPluginUI::Create.

If you must display a dialog box or message box from a UI plug-in method implementation that is called by the Player, you must follow these steps:

1. Register a new window message using RegisterWindowMessage.
2. Send the window message you registered to your UI plug-in window using PostMessage.
3. Show the dialog box or message box from the message handler for your new window message.

See Also

- UI Plug-in Overview
Building a UI Plug-in

Writing your own UI plug-in is easy if you understand Windows programming and have some familiarity with the Microsoft Component Object Model (COM). Knowledge of the Active Template Library (ATL) is also helpful.

UI Plug-ins are COM controls that are packaged as dynamic link libraries (DLLs). These controls must be single-threaded apartment (STA) plug-in COM objects, and must be registered in the Windows registry to be used. Windows Media Player must be notified during plug-in installation that the plug-in is available. When this happens, the plug-in will load automatically if it has been configured to do so, and will appear in the Plug-ins list on the Tools menu.

The Windows Media Player SDK provides wizards for Microsoft Visual C++ 6.0 and Microsoft Visual C++ .NET that take care of these standard implementation details. In order to create a basic UI plug-in using a wizard, all that is required is the implementation of a user interface and the modification of a few IWMPPluginUI interface methods. Windows Media Player provides the plug-in with access to its core functionality through the IWMPCore interface, and provides a parent window for whatever user interface controls the plug-in requires.

The following sections explain what you need to know to create a UI plug-in using the wizard:

- Getting Started with the UI Plug-in Wizard
- Using the UI Plug-in Wizard with Visual C++ 6.0
- Using the UI Plug-in Wizard with Visual C++ .NET
- Compiling the UI Plug-in Project
- Customizing the UI Plug-in

See Also

- About User Interface Plug-ins
Getting Started with the UI Plug-in Wizard

In order to develop a UI plug-in, you must install the following tools:

- Microsoft Visual C++ 6.0 or later.
- Windows Media Player 9 Series or later
- Windows Media Player 9 Series SDK or later
- Windows Media Player UI Plug-in Wizard

Installing the Wizard

There are separate versions of the Windows Media Player Plug-in Wizard for Microsoft Visual C++ 6.0 and Microsoft Visual C++ .NET.

Once you have installed the Windows Media Player SDK, you can find the UI Plug-in Wizard for Visual C++ 6.0 in the following directory:

\wizards\wmpplugin

The wizard file is named wmpwiz.awx.

Copy the wizard file to the following directory on your hard disk:

\Program Files\Microsoft Visual Studio\Common\MSDev98\Template

If you installed Microsoft Visual Studio 6.0 (which contains Microsoft Visual C++ 6.0) in another directory, find the corresponding directory. If you are not sure where you installed it, you can search for other COM wizards with the .awx file name extension.

If you are using Microsoft Visual C++ .NET, the Windows Media Player plug-in wizard is installed and configured automatically for you when you install this SDK. If you install Visual C++ .NET after installing this SDK, you should remove the SDK and reinstall it.

See Also

- Building a UI Plug-in

© 2000-2003 Microsoft Corporation. All rights reserved.

Using the UI Plug-in Wizard with Visual C++ 6.0
Once you have installed the necessary components, creating a sample UI plug-in is easy. The following steps will guide you:

2. From the File menu, click New.
4. In the Project name box, type a name for your UI plug-in.
5. In the Location box, provide the location of a folder where the wizard can save the files it generates. You can accept the default, type a new path, or browse to an existing location.
6. Click OK.
7. Click UI Plugin. Click Next.
8. The wizard displays a list of UI plug-in types to choose from. Once you make your choice, click Next.
9. Provide a Friendly name and Description for your plug-in.
10. Specify various features to be included with your plug-in. These include listening to Windows Media Player events, automatically loading the plug-in when it is installed, and providing a property page with the option of automatic display the first time the plug-in is loaded.
11. Click Finish. You will be shown a dialog box containing information about your new project. When you close this dialog box by clicking OK, the wizard will generate the source files needed to compile your project.

The wizard generates a complete UI plug-in project.

See Also

- Building a UI Plug-in
10. Provide a **Friendly name** and **Description** for your plug-in.
11. Specify various features to be included with your plug-in. These include listening to Windows Media Player events, automatically loading the plug-in when it is installed, and providing a property page with the option of automatic display the first time the plug-in is loaded.
12. Click **Next**.

The wizard generates a complete UI plug-in project.

**See Also**

- [Building a UI Plug-in](#)

---

**Compiling the UI Plug-in Project**

Your project is now ready to compile. If you build the project now, Visual C++ will create the plug-in DLL file and register it in the Windows registry. When you launch Windows Media Player, the plug-in will be available for use, and will load automatically if it has been configured to do so.

The plug-in code that the wizard generates does just enough to demonstrate that it works by displaying its title in the appropriate location for the specified plug-in type. If the plug-in has a property page, the text that is displayed can be modified in the property page text box. If it is a background plug-in, the background plug-in icon will appear next to the running track time display.

**See Also**

- [Building a UI Plug-in](#)
Customizing the UI Plug-in

At this point, your project is ready for customization. You can modify the wizard-generated implementation of the IWMPPluginUI interface, you can add a user interface to the CPluginWindow class, and you can implement a property page in the CPropertyDialog class. If your plug-in is configured to listen to Windows Media Player events, the wizard will have generated default or empty implementations of all the necessary event handlers, which you also modify or create.

The type of plug-in and the features it supports are indicated by a value which is stored in the Windows registry. The wizard generates a file with a .rgs file name extension that contains the information to register the plug-in with. The Capabilities value in this file is the decimal equivalent of a Boolean OR of the plug-in type constants and plug-in flags defined in wmpplug.idl. Although this value is determined by the options you select in the wizard, you must modify it if you want to create a plug-in with multiple presets or one that media items or playlists can be sent to.

As you modify and extend your plug-in code, you can build and register your DLL in order to test your plug-in in Windows Media Player.

See Also

- Building a UI Plug-in

User Interface Plug-ins Programming Guide

The code sample described in this section demonstrates the process of implementing a custom UI plug-in starting with code generated by the Windows Media Player Plug-in Wizard.

The Search UI plug-in is a metadata area plug-in that provides a Search button. When this button is clicked, a search page is launched in the default Web browser that contains information about the artist of the current media item.

The first step in creating this plug-in is to start a new project in Microsoft Visual C++ by selecting Windows Media Player Plug-in Wizard from the Projects tab. Name the project "Search", and click OK. Choose UI Plug-in and click Next. Then choose the Metadata type from the options list and click Next. Finally, click the check box for auto-run support so that the plug-in will load automatically, and then click Finish. The wizard generates the required project files, including basic implementations of the CSearch class and the IWMPPluginUI interface that it supports, and the CPluginWindow class that provides the user interface. This is the code that will be modified to provide the plug-in functionality described in this section.
This section contains the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementing CSearch</td>
<td>Describes the changes required to the CSearch class.</td>
</tr>
<tr>
<td>Implementing CPluginWindow</td>
<td>Describes the changes required to the CPluginWindow class.</td>
</tr>
</tbody>
</table>

See Also

- User Interface Plug-ins Programming Guide

Implementing CSearch

The IWMPPPluginUI interface has several methods that are called by Windows Media Player at different times during the life cycle of a plug-in instance. The wizard provides basic implementations of these methods as well as the class constructor and destructor and other class methods. The Search.h file must be modified so that Windows Media Player can communicate with the user interface, which is described in the next section.

In order for the CPluginWindow class to have access to the private member variable m_spCore, a friend class declaration must be made inside the CSearch class definition as shown in the following code snippet:

```cpp
class ATL_NO_VTABLE CSearch :
   public CComObjectRootEx<CComSingleThreadModel>,
   public CComCoClass<CSearch, &CLSID_Search>,
   public IWMPPPluginUI
{
   friend class CPluginWindow;

   // Rest of class definition...
}
```

See Also

- User Interface Plug-ins Programming Guide
Implementing CPluginWindow

The CPluginWindow class provides the user interface to the plug-in. In the case of the Search UI plug-in, this class contains the code for the Search button and the code that launches the search page when the button is clicked.

The wizard provides a basic implementation of CPluginWindow in the CPluginWindow.h header file. To keep things simple, the Search UI plug-in will modify this file directly, although extensive additions would normally be placed in a separate CPluginWindow.cpp file.

The following sections describe what you need to do to implement CPluginWindow:

- The Message Map
- The Constructor
- The OnPaint Method
- The OnCreate Method
- The OnSearch Method
- The LaunchPage Method

See Also

- User Interface Plug-ins Programming Guide

The Message Map

The plug-in window responds to various events by calling methods that are mapped to corresponding event messages. The wizard provides a mapping so that OnPaint and OnEraseBackground will be called at the appropriate times. In order to create the Search button and to respond to clicks from it, the message map section is modified as follows:
BEGIN_MSG_MAP(CPluginWindow)
    MESSAGE_HANDLER(WM_PAINT, OnPaint)
    MESSAGE_HANDLER(WM_ERASEBKGND, OnEraseBackground)
    MESSAGE_HANDLER(WM_CREATE, OnCreate)
    COMMAND_ID_HANDLER(IDC_SEARCH, OnSearch)
END_MSG_MAP()

See Also

- Implementing CPluginWindow

---

**The Constructor**

The constructor that the wizard provides stores a pointer to the main plug-in class in a member variable. The wizard uses this reference to gain access to member variables declared in the CSearch class. No implementation change is needed.

See Also

- Implementing CPluginWindow

---

**The OnPaint Method**

The OnPaint method is called whenever the plug-in window should paint itself. This occurs when the plug-in window receives a WM_PAINT message, which is mapped to the OnPaint method in the message map described earlier. The wizard provides an implementation of this method that paints the background black and places the name of the plug-in in the plug-in window. The only modification that is necessary for the Search UI plug-in is the removal of the code that displays the text.
The following code is used to implement this method:

```c
LRESULT OnPaint(UINT nMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
{
    PAINTSTRUCT ps;
    HDC hDC = BeginPaint(&ps);
    RECT rc;
    GetClientRect(&rc);
    HBRUSH hNewBrush = ::CreateSolidBrush( RGB(0, 0, 0) );
    if (hNewBrush)
    {
        ::FillRect(hDC, &rc, hNewBrush );
        ::DeleteObject( hNewBrush );
    }
    EndPaint(&ps);
    return 0;
}
```

See Also

- Implementing CPluginWindow

The OnCreate Method

The OnCreate method is called when the plug-in window is first created.

The following code is used to implement this method:

```c
LRESULT OnCreate(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
{
    HWND hCtrl = ::CreateWindowEx(0L, _T("BUTTON"), _T("Search"),
                                    WS_CHILD | BS_PUSHBUTTON, 10, 10, 100, 30, m_hWnd,
                                    (HMENU)IDC_SEARCH, _Module.GetResourceInstance(), NULL);
    ::ShowWindow(hCtrl, SW_SHOW );
    return 0;
}
```

This method creates the **Search** button and associates it with the IDC_SEARCH command ID, which is defined
at the beginning of the file:

#define IDC_SEARCH 2000

This command ID is mapped to the OnSearch method in the message map section described previously.

See Also

- Implementing CPluginWindow

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

The OnSearch Method

The OnSearch method is called by Windows Media Player when the Search button is clicked. This method retrieves the current Media object and passes it to the LaunchPage method.

The following code is used to implement this method:

```cpp
LRESULT OnSearch(WORD wNotifyCode, WORD wID, HWND hwndCtl, BOOL& fHandled)
{
    HRESULT hr;
    CComPtr<IWMPMedia> spMedia;

    if ( m_pPlugin && m_pPlugin->m_spCore )
    {
        // Get a pointer to the current media item.
        hr = m_pPlugin->m_spCore->get_currentMedia(&spMedia);
        if (SUCCEEDED(hr) && spMedia)
        {
            LaunchPage(spMedia);
        }
        else
        {
            MessageBox(_T("There is no media loaded."), _T("Warn"), MB_OK | MB_ICONWARNING
        }
    }
    return 0;
}
```

See Also

- Implementing CPluginWindow
The LaunchPage Method

The LaunchPage method provides the primary functionality of the plug-in, which is to launch a search page containing information about the artist of the media item passed to the method.

This method is called by the OnSearch method using the current Media object.

The following code is used to implement this method:

```cpp
void LaunchPage(IWMPMedia *pMedia)
{
    USES_CONVERSION;

    HRESULT hr;
    CComBSTR bstrType;
    CComBSTR bstrArtist;

    // Get the name of the artist.
    bstrType = _T("artist");
    hr = pMedia->getItemInfo(bstrType, &bstrArtist);
    if (SUCCEEDED(hr))
    {
        // Create the search URL.
        TCHAR szSearch[MAX_PATH];
        _stprintf(szSearch, _T("http://search.msn.com/results.asp?q=%s"), OLE2T(bstrArtist));
        CComBSTR bstrURL = szSearch;

        // Launch the search page.
        m_pPlugin->m_spCore->launchURL(bstrURL);
    }
    else
    {
        MessageBox(_T("Failed to get artist information from media."), _T("Warn"), MB_OK |
    }
}
```

See Also

- Implementing CPluginWindow
User Interface Plug-ins Programming Reference

The Microsoft Windows Media Player Software Development Kit (SDK) supports one interface for UI plug-ins. The following section documents this in detail.

Section

WMPNotifyPluginAddRemove

Description

An independent function used to notify Windows Media Player that a plug-in has been installed or uninstalled.

IWMPPluginUI Interface

Description

An interface to UI plug-ins.

Registration Flags

Description

A list of settings flags that affect the behavior of UI plug-ins.

See Also

- Windows Media Player User Interface Plug-ins

WMPNotifyPluginAddRemove

The WMPNotifyPluginAddRemove function notifies Windows Media Player that a plug-in has been installed or uninstalled.

Syntax

BOOL WMPNotifyPluginAddRemove();

Parameters

This function takes no parameters.
Return Values

This method returns a `BOOL` value that indicates whether the function call succeeded.

Remarks

This function is typically called by a user interface (UI) plug-in in its DllRegisterServer and DllUnregisterServer methods. **Windows Media Player Plug-in Wizard** generates this code automatically, so it is only necessary to add calls to this function to UI plug-ins created without the wizard.

Requirements

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpplug.idl; include wmpplug.h.

See Also

- [User Interface Plug-ins Programming Reference](#)

Registration Flags

When the Windows Media Player Plug-in Wizard creates a new UI plug-in project, it creates a key in the registry that contains information about the plug-in. This key is created in the following location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MediaPlayer\UIPlugins\{ClassId}
```

*ClassId* is the class id of the plug-in.

This key includes the following values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capabilities</td>
<td>REG_DWORD</td>
<td>A DWORD value that consists of at least one plug-in type flag that may be combined with one or more plug-in capabilities flags by using binary OR operations.</td>
</tr>
</tbody>
</table>
For more information about the res protocol, see the Internet Development SDK.

The following table details the plug-in type flags.

<table>
<thead>
<tr>
<th>Plug-in Type Flag</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUGIN_TYPE_BACKGROUND</td>
<td>0x1</td>
<td>The UI plug-in does not display a user interface.</td>
</tr>
<tr>
<td>PLUGIN_TYPE_SEPARATEWINDOW</td>
<td>0x20x2</td>
<td>The UI plug-in is a separate window plug-in.</td>
</tr>
<tr>
<td>PLUGIN_TYPE_DISPLAYAREA</td>
<td>0x3</td>
<td>The UI plug-in is a display area plug-in.</td>
</tr>
<tr>
<td>PLUGIN_TYPE_SETTINGSAREA</td>
<td>0x4</td>
<td>The UI plug-in is a settings area plug-in.</td>
</tr>
<tr>
<td>PLUGIN_TYPE_METADATAAREA</td>
<td>0x5</td>
<td>The UI plug-in is a metadata area plug-in.</td>
</tr>
</tbody>
</table>

The following table details the plug-in capabilities flags.

<table>
<thead>
<tr>
<th>Plug-in Capabilities Flag</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUGIN_FLAGS_ACCEPTS_MEDIA</td>
<td>0x10000000</td>
<td>The UI plug-in can accept Media object pointer arrays when Windows Media Player calls IWMPPPlugin::SetProperty.</td>
</tr>
<tr>
<td>PLUGIN_FLAGS_ACCEPTS_PLAYLISTS</td>
<td>0x80000000</td>
<td>The UI plug-in can accept Playlist object pointer arrays when Windows Media Player calls IWMPPPluginUI::SetProperty.</td>
</tr>
</tbody>
</table>
| PLUGIN_FLAGS_HAS_PRESETS          | 0x40000000  | The UI plug-in uses presets. If the plug-in specifies this flag, Windows Media Player will query the plug-in for preset informatio
The following constants are defined in wmpplug.h. Do not change the values associated with these constants.

**PLUGIN_FLAGS_HASPROPERTYPAGE** 0x80000000 The UI plug-in provides a property page dialog. Windows Media Player will call **WMPPluginUI::DisplayPropertyPage** if this flag is set when the property page is invoked.

**PLUGIN_FLAGS_HIDDEN** 0x02000000 The background UI plug-in does not appear on the **Plug-ins** menu that is accessed from the **View** or **Tools** menus or the **Select Now Playing options** button in Now Playing. It does appear on the **Plug-ins** tab of the Options dialog. It does cause the Background Plug-in Running icon to appear in the status bar.

This flag has no effect on plug-ins other than background UI plug-ins.

**PLUGIN_FLAGS_INSTALLAUTORUN** 0x40000000 Windows Media Player runs the UI plug-in automatically when the plug-in is installed.

**PLUGIN_FLAGS_LAUNCHPROPERTYPAGE** 0x20000000 Windows Media Player calls **WMPPluginUI::DisplayPropertyPage** when the UI plug-in runs for the first time.

If this flag is specified, **PLUGIN_FLAGS_HASPROPERTYPAGE** should be specified also.

The following constants are defined in wmpplug.h. Do not change the values associated with these constants.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUGIN_INSTALLREGKEY</td>
<td>The location of the plug-in registry key.</td>
</tr>
<tr>
<td>PLUGIN_INSTALLREGKEY_FRIENDLYNAME</td>
<td>The name of the friendly name value.</td>
</tr>
<tr>
<td>PLUGIN_INSTALLREGKEY_DESCRIPTION</td>
<td>The name of the description value.</td>
</tr>
<tr>
<td>PLUGIN_INSTALLREGKEY_CAPABILITIES</td>
<td>The name of the capabilities value.</td>
</tr>
<tr>
<td>PLUGIN_INSTALLREGKEY_UNINSTALL</td>
<td>The name of the uninstall path value.</td>
</tr>
</tbody>
</table>

**See Also**

- **WMPPluginUI::DisplayPropertyPage**
- **WMPPluginUI::GetProperty**
- **WMPPluginUI::SetProperty**
- User Interface Plug-ins Programming Reference
### IWMPPluginUI Interface

The **IWMPPluginUI** interface manages the connection to Windows Media Player.

In addition to the methods inherited from **IUnknown**, the **IWMPPluginUI** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>Called by Windows Media Player to instantiate the plug-in user interface.</td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td>Called by Windows Media Player to shut down the plug-in user interface.</td>
</tr>
<tr>
<td><strong>DisplayPropertyPage</strong></td>
<td>Called by Windows Media Player to request that the plug-in display its property page.</td>
</tr>
<tr>
<td><strong>GetProperty</strong></td>
<td>Called by Windows Media Player to retrieve name/value property pairs from the plug-in.</td>
</tr>
<tr>
<td><strong>SetCore</strong></td>
<td>Called by Windows Media Player to provide plug-in access to the core Windows Media Player APIs.</td>
</tr>
<tr>
<td><strong>SetProperty</strong></td>
<td>Called by Windows Media Player to set name/value property pairs for the plug-in.</td>
</tr>
<tr>
<td><strong>TranslateAccelerator</strong></td>
<td>Called as part of the Windows Media Player message loop to allow the plug-in to intercept and respond to keyboard events.</td>
</tr>
</tbody>
</table>
IWMPPluginUI::Create

The Create method is called by Windows Media Player to instantiate the plug-in user interface. This method is passed a handle to a parent window of the plug-in window. A handle to the newly created window is then passed back to the calling method.

Syntax

```c
HRESULT Create(
    HWND hwndParent,
    HWND* phwndWindow
);
```

Parameters

hwndParent

[in] HWND handle to a parent window of the plug-in window.

phwndWindow

[out] Pointer to an HWND handle to the plug-in window after the content is filled in.

Return Values

This method returns an HRESULT.

Remarks

This method is called by Windows Media Player for all user interface plug-in types except the background type.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpplug.idl; include wmpplug.h.

See Also

- IWMPPluginUI Interface
IWMPPluginUI::Destroy

The **Destroy** method is called by Windows Media Player to shut down the plug-in user interface. This occurs when the user closes a plug-in in a separate window, switches out of the **Now Playing** pane, or selects a different display, settings, or metadata area plug-in to display in the **Now Playing** pane.

**Syntax**

`HRESULT Destroy();`

**Parameters**

This method takes no parameters.

**Return Values**

This method returns an **HRESULT**.

**Remarks**

This method is called by Windows Media Player for all user interface plug-in types except the background type.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpplug.idl; include wmpplug.h.

**See Also**

- [IWMPPluginUI Interface](#)

IWMPPluginUI::DisplayPropertyPage

The **DisplayPropertyPage** method is called by Windows Media Player to request that the plug-in display its property page. This method is passed a handle to a parent window of the plug-in property page dialog box.
Syntax

HRESULT DisplayPropertyPage(
    HWND hwndParent
);

Parameters

hwndParent

[in] HWND handle to a parent window of the property page dialog box.

Return Values

This method returns an HRESULT.

Remarks

This method is called by Windows Media Player only for plug-ins that provide a property page.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpplug.idl; include wmpplug.h.

See Also

- IWMPPluginUI Interface

IWMPPluginUI::GetProperty

The GetProperty method is called by Windows Media Player to retrieve name/value property pairs from the plug-in.

Syntax

HRESULT GetProperty(
    const WCHAR* pswzName,
    VARIANT* pvarProperty
)
Parameters

\textit{pswzName}

[in] Pointer to a \texttt{WCHAR} NULL-terminated string constant containing the name of the property. Contains one of the following values:

\begin{tabular}{|l|l|}
\hline
\textbf{Value} & \textbf{Description} \\
\hline
PLUGIN_MISC_CURRENTPRESET = L"CurrentPreset" & The out parameter is set to a \texttt{Long (VT\_I4)} value containing the index of the current preset. This property is requested only for plug-ins that have presets. \\
\hline
PLUGIN_MISC_PRESETCOUNT = L"PresetCount" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the number of presets available in the plug-in. This property is requested only for plug-ins that have presets. \\
\hline
PLUGIN_MISC_PRESETNAMES = L"PresetNames" & The out parameter is set to an array of \texttt{BSTR (VT\_ARRAY | BSTR)} values containing the names of the presets. This property is requested only for plug-ins that have presets. \\
\hline
PLUGIN_MISC_QUERYDESTROY = L"QueryDestroy" & The out parameter is set to a \texttt{BSTR (VT\_BSTR)} value that is displayed to the user when Windows Media Player attempts to close a separate window or background plug-in that is engaged in operations that cannot be interrupted. \\
\hline
PLUGIN_SEPARATEWINDOW_DEFAULTHEIGHT = L"DefaultHeight" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the desired default opening height of the plug-in window. This property is requested only for plug-ins in separate windows. \\
\hline
PLUGIN_SEPARATEWINDOW_DEFAULTWIDTH = L"DefaultWidth" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the desired default opening width of the plug-in window. This property is requested only for plug-ins in separate windows. \\
\hline
PLUGIN_SEPARATEWINDOW_MAXHEIGHT = L"MaxHeight" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the desired maximum height of the plug-in window. This property is requested only for plug-ins in separate, resizable windows. \\
\hline
PLUGIN_SEPARATEWINDOW_MAXWIDTH = L"MaxWidth" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the desired maximum width of the plug-in window. This property is requested only for plug-ins in separate, resizable windows. \\
\hline
PLUGIN_SEPARATEWINDOW_MINHEIGHT = L"MinHeight" & The out parameter is set to a \texttt{Long (VT\_I4)} value indicating the desired minimum height of the plug-in window. This property is requested only for plug-ins in separate, resizable windows. \\
\hline
\end{tabular}
pvarProperty

[out] Pointer to a VARIANT to contain the value of the property.

Return Values

This method returns an HRESULT.

Remarks

Windows Media Player determines the type and capabilities of a plug-in by checking the Windows registry, and will retrieve only properties that the plug-in supports.

When a user attempts to close a separate window or background UI plug-in, or to close Windows Media Player when one of these plug-in types is active, this method is called with the PLUGIN_MISC_QUERYDESTROY property specified. If the plug-in is engaged in an operation that cannot be interrupted, such as reading or writing a file or waiting for user input in a modal dialog box, set the out parameter of this method to a non-empty value. This value is displayed to the user to indicate the problem. A user who is attempting to close Windows Media Player is then given the option of overriding the plug-in and closing the Player anyway.

When the plug-in is ready to close, set the out parameter to "" (empty string). When Windows Media Player calls this method and receives an empty value in the out parameter, it closes the plug-in without further delay.

This method is not called when a display area, settings area, or metadata area plug-in is closed. Because these plug-in types are displayed in the Now Playing pane, they must be ready to close at any time, such as when a user switches to another pane.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpplug.idl; include wmpplug.h.

See Also

- IWMPPluginUI Interface

© 2000-2003 Microsoft Corporation. All rights reserved.
IWMPluginUI::SetCore

The SetCore method is called by Windows Media Player to provide plug-in access to the core Windows Media Player APIs.

Syntax

HRESULT SetCore(
    IWMPCore* pCore
);

Parameters

pCore

[in] Pointer to an IWMPCore interface.

Return Values

This method returns an HRESULT.

Remarks

This method is called by Windows Media Player to allow the plug-in to set or release a pointer to the IWMPCore interface. If pCore is NULL, the plug-in is being shut down and all stored references to the core should be released.

This method is not called when Windows Media Player instantiates the plug-in for the purpose of displaying its property page. This method can therefore be used as an entry point that will only be called when the plug-in is enabled and Windows Media Player loads it normally.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpplug.idl; include wmpplug.h.

See Also

- IWMPluginUI Interface
The `SetProperty` method is called by Windows Media Player to set name/value property pairs for the plug-in.

**Syntax**

```cpp
HRESULT SetProperty(
    const WCHAR* pswzName,
    VARIANT* pvarProperty
);
```

**Parameters**

- `pswzName`  
  [in] Pointer to a WCHAR NULL-terminated string constant containing the name of the property. Contains one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUGIN_MISC_CURRENTPRESET = L&quot;CurrentPreset&quot;</td>
<td>The <code>pvarProperty</code> parameter contains a <strong>Long</strong> (VT_I4) value that specifies the index of the plug-in preset which is to be made current.</td>
</tr>
<tr>
<td>PLUGIN_ALL_MEDIASENDT0 = L&quot;MediaSendTo&quot;</td>
<td>The <code>pvarProperty</code> parameter contains an array of <strong>IUnknown</strong> (VT_ARRAY</td>
</tr>
<tr>
<td>PLUGIN_ALL_PLAYLISTSENDT0 = L&quot;PlaylistSendTo&quot;</td>
<td>The <code>pvarProperty</code> parameter contains an array of <strong>IUnknown</strong> (VT_ARRAY</td>
</tr>
</tbody>
</table>

- `pvarProperty`  
  [in] Pointer to a VARIANT containing the new value of the property.

**Return Values**

This method returns an **HRESULT**.

**Remarks**

Windows Media Player determines the type and capabilities of a plug-in by checking the Windows registry, and
will specify only properties that the plug-in supports.

**Media** and **Playlist** objects are sent to the plug-in as arrays of **IUnknown** pointers. The plug-in can call **QueryInterface** on these pointers to retrieve **IWMPMedia** or **IWMPPlaylist** interfaces.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpplug.idl; include wmpplug.h.

**See Also**

- **IWMPPluginUI Interface**
- **Media Object**

---

**IWMPPluginUI::TranslateAccelerator**

The **TranslateAccelerator** method is called as part of the Windows Media Player message loop to allow the plug-in to intercept and respond to keyboard events.

**Syntax**

```cpp
HRESULT TranslateAccelerator(
    LPMSG lpmsg
);
```

**Parameters**

*lpmsg*

[in] **LPMSG** structure containing message information from Windows Media Player that the plug-in can respond to.

**Return Values**

This method returns an **HRESULT**.

**Remarks**
The plug-in can set up an accelerator table to reroute specific keyboard events to appropriate handler methods. If the plug-in chooses not to respond to keyboard events, it should return **S_FALSE**.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpplug.idl; include wmpplug.h.

See Also

- **IWMPPPluginUI Interface**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Windows Media Player DSP Plug-ins**

Microsoft Windows Media Player provides an architecture that enables the user to install and activate plug-in programs that add digital signal processing (DSP) functionality. DSP plug-ins are Microsoft DirectX Media Objects (DMOs) that connect to the Player by using COM interfaces. A typical DSP plug-in might be an audio equalizer or a video tint control. This section of the SDK provides the programming information you need to create your own DSP plug-in.

The DSP plug-in documentation is divided into three sections:

**Section**

**About DSP Plug-ins**

Provides an overview of the architecture used for DSP plug-ins. Read this section to learn the general concepts involved with this technology.

**DSP Plug-ins Programming Guide**

Explains what you need to do to create a DSP plug-in. This section contains example code and step-by-step procedures.

**DSP Plug-ins Programming Reference**

Provides a detailed reference for the COM interfaces, methods, and enumerated types supported by the Windows Media Player SDK for DSP plug-ins.

See Also

- **Windows Media Player Plug-ins**
About DSP Plug-ins

Microsoft Windows Media Player provides architecture for connecting add-on software that performs digital signal processing (DSP) functions on the audio or video streams during playback. These add-on programs are referred to as DSP plug-ins.

The following sections provide a conceptual overview of Windows Media Player DSP plug-ins.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP Plug-in User Overview</td>
<td>Describes DSP plug-ins from the end user's perspective.</td>
</tr>
<tr>
<td>DSP Plug-in Developer Overview</td>
<td>Provides general information you need to know before you build a DSP plug-in for Windows Media Player.</td>
</tr>
<tr>
<td>Building a DSP Plug-in</td>
<td>Explains how to use the Windows Media Player plug-in wizard to create a sample DSP plug-in.</td>
</tr>
<tr>
<td>Implementing Your DSP Code</td>
<td>Discusses considerations for creating custom audio and video DSP plug-ins for Windows Media Player.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player DSP Plug-ins
DSP Plug-in User Overview

To the user, DSP plug-ins are add-on features that somehow change the way the user experiences the audio or video that Windows Media Player is playing. The result of a DSP plug-in could be something obvious, like the colors in the video image being reversed so it looks like a photographic negative, or something very subtle, like a light reverb effect in the audio to add a feeling of spaciousness to the sound.

Users can install and uninstall DSP plug-ins by using the Plug-ins tab of the Options dialog box in Windows Media Player. Users can then enable and disable installed plug-ins using the checkboxes provided on the Plug-ins tab. DSP plug-ins may implement a property page, which users can also access from the Plug-ins tab. A property page provides the user with a way to change settings that affect the way a DSP plug-in works. For example, a property page might include a slider control that allows the user to change the decay time of a reverb effect.

See Also

- About DSP Plug-ins

DSP Plug-in Developer Overview

From the developer's perspective, DSP plug-ins are software programs that receive audio or video data provided by Windows Media Player just before the rendering stage, then process that data and return the data to Windows Media Player for rendering.

The following sections provide general information about Windows Media Player DSP plug-ins:

- DSP Plug-in Packaging
- Connecting to Windows Media Player
- Data Input and Output
- Format Negotiation
- Providing a User Interface
- DSP Plug-in Wizard
- Required Interfaces
- Registering DSP Plug-ins

See Also

- About DSP Plug-ins
DSP Plug-in Packaging

Windows Media Player DSP plug-ins are based upon Microsoft DirectXT Media Objects (DMOs). DMOs are programming objects built using Microsoft Component Object Model (COM) technology. They are designed to provide a lightweight container for audio or video processing algorithms that can be used by a variety of digital media applications. It is common to think of a DMO as being analogous to a filter object in Microsoft DirectShow®. In fact, a DMO can be inserted into a DirectShow filter graph by using a DMO Wrapper filter.

The DMO documentation can be found in the DirectShow section of the Microsoft DirectX 8 Software Development Kit (SDK) or any later version of the SDK. If you plan to create your own DSP plug-ins for Windows Media Player, you should install the entire DirectX SDK because it includes header and library files that you will need. You can download the DirectX SDK from the Microsoft Web site.

Windows Media Player DSP plug-ins are packaged and distributed as self-registering .dll files.

DSP plug-in objects must not be created as singletons. Windows Media Player must be able to create multiple separate instances of a particular DSP plug-in object.

See Also

- DSP Plug-in Developer Overview

Connecting to Windows Media Player

Windows Media Player automatically connects to DSP plug-ins that have been installed and properly registered. DSP plug-ins must call IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin to create the registry entries necessary to allow Windows Media Player to recognize the plug-in and to list it on the Plug-ins tab of
the Options dialog box. To remove the registry entries created by \nIWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin, the plug-in calls \nIWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin.

See Also

- DSP Plug-in Developer Overview

Data Input and Output

Windows Media Player provides audio or video data to DSP plug-ins through an input buffer allocated by Windows Media Player. DSP plug-ins return processed data to Windows Media Player through an output buffer that is also allocated by Windows Media Player. Windows Media Player manages the process of passing data between itself and the DSP plug-in by calling methods implemented by the plug-in. The process works as follows:

1. Windows Media Player calls IMediaObject::ProcessInput, passing a pointer to an IMediaBuffer object to the DSP plug-in.
2. The DSP plug-in keeps a reference count on the input buffer object. The DSP plug-in returns an appropriate success or failure HRESULT.
3. Windows Media Player calls IMediaObject::ProcessOutput, passing a pointer to an array of DMO_OUTPUT_DATA_BUFFER structures (which contain output buffers) to the DSP plug-in.
4. The DSP plug-in processes the data in the input buffer and then copies the data to the appropriate output buffer. The DSP plug-in releases the reference count on the input buffer object when all the data in the buffer has been processed. The DSP plug-in then returns an appropriate success or failure HRESULT.

Note Do not write code that writes data to the input buffer or reads data from the output buffer. Incorrectly accessing data buffers may yield unexpected results.

5. Windows Media Player renders the content in the output buffer.

This process repeats continuously while the plug-in is enabled and Windows Media Player has content to render.

See Also

- DSP Plug-in Developer Overview
Format Negotiation

In order for Windows Media Player and a DSP plug-in to be able to share data, both programs must agree on the data format they are processing. Windows Media Player only plays content that exists in a format that it can process, but it needs a way to verify that a DSP plug-in supports the format of the current media before it can provide data to the plug-in. The IMediaObject interface provides methods that a DSP plug-in implements in order to provide Windows Media Player with information about the types of media supported by the plug-in and to enable Windows Media Player to specify media types for input and output. These methods use the DMO_MEDIA_TYPE structure, defined in mediaobj.h, to identify particular media types. For more information about format negotiation, see About IMediaObject.

DSP plug-ins can process audio of a variety of bit depths. In general, Windows Media Player will only load DSP plug-ins that can process the same bit depth as the digital audio. For instance, if the digital audio is 20-bit, the plug-in must be written to process 20-bit audio. For CD audio, DSP plug-ins must support 20-bit processing.

During format negotiation of multi-channel content on a computer configured for use with stereo speakers, Windows Media Player first attempts to connect to an audio DSP plug-in using the existing input and output format by calling IMediaObject::SetInputType and IMediaObject::SetOutputType. Once this initial negotiation occurs, the Player then enumerates the formats the plug-in supports and attempts to negotiate the best format combination for the Player and the plug-in. If the plug-in accepts stereo audio (defined by a WAVEFORMATEX structure) as the input format during the initial negotiation, and then subsequently accepts only multi-channel audio (defined by a WAVEFORMATEXTENSIBLE structure), the Player will provide multi-channel audio as the input format to the plug-in. This behavior during format negotiation is available for use in the Microsoft Windows XP operating system. It may be altered or unavailable in subsequent versions.

See Also

- DSP Plug-in Developer Overview
Providing a User Interface

DSP plug-ins can provide a property page in order to create a user interface. To do this, the plug-in must include a property page object that provides an implementation of an `IPropertyPage` interface. The DSP plug-in object must implement `ISpecifyPropertyPages::GetPages`, which allows Windows Media Player to locate and identify the correct property page for the plug-in.

Displaying a Status Graphic

DSP plug-ins can display a small graphic, or series of graphics, in the Windows Media Player status area to notify the user that a plug-in is active. To support this feature, the plug-in must implement the `IPropertyBag` interface. Windows Media Player calls `IPropertyBag::Read`, providing a pointer to the requested property name "IconStreams", which is case-sensitive, and a pointer to a VARIANT structure that receives the data for the graphic. The plug-in creates an `IStream` object (or a SAFEARRAY of `IStream` objects if there are multiple graphics), then loads the graphic data, including header information, into the stream, and then returns a pointer to the `IStream` object using the `punkVal` member of the VARIANT structure. If the plug-in only supplies one graphic, it specifies the `vt` member of the VARIANT structure as VT_UNKNOWN. If the plug-in supplies multiple graphic `IStream` objects using a SAFEARRAY, it specifies the `vt` member of the VARIANT structure as VT_ARRAY.

Graphics can be stored in a variety of file formats, including:

**BMP**

Microsoft Windows Bitmap images are uncompressed.

**JPEG**

Compressed image format commonly used for Web pages. JPEG format files usually have .jpg file name extensions.

**GIF**

Compressed image format commonly used for Web pages.

**PNG**

Compressed image format commonly used for Web pages.

The maximum dimensions for DSP plug-in graphics are 38 pixels wide and 14 pixels high.

The `IStream` byte stream containing the status graphic must include header information. Without header information, Windows Media Player cannot properly identify the type of graphic and therefore will not load the image.

See Also

- DSP Plug-in Developer Overview
DSP Plug-in Wizard

The Windows Media Player SDK provides a COM wizard that you can add to Visual C++ 6.0 or Visual C++ .NET that will generate the code for a DSP plug-in. This code includes sample method implementations that scale the volume of the audio. You can compile and link this sample code and register the resulting DLL file to try the sample DSP plug-in. You can then modify the generated code to create your own DSP plug-in. For more information, see Building a DSP Plug-in.

See Also

- DSP Plug-in Developer Overview

Required Interfaces

A Windows Media Player DSP plug-in must implement the IMediaObject interface and the IWMPPluginEnable interface as a minimum requirement. The following sections provide an overview of these interfaces.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About IMediaObject</td>
<td>This is also the required interface for a DirectX Media Object (DMO). Detailed documentation for this interface is located in the DirectX 8 SDK.</td>
</tr>
<tr>
<td>About IWMPPluginEnable</td>
<td>This interface stores a value indicating whether Windows Media Player has enabled the plug-in.</td>
</tr>
</tbody>
</table>

In addition to these interfaces, the plug-in can implement any additional interfaces required to do the job. For instance, DMOs can implement several interfaces specific to the DMO architecture. By implementing the DMO
interfaces as required, you can create a plug-in that functions both as a DMO and as a Windows Media Player plug-in.

See Also

- DSP Plug-in Developer Overview
- DSP Plug-in Interfaces

About IMediaObject

The **IMediaObject** interface is the required interface for DMOs. **IMediaObject** contains the methods that a Windows Media Player DSP plug-in uses to get data from Windows Media Player, to process the data, and to return the processed data to Windows Media Player. For complete documentation of the **IMediaObject** interface, see the DirectX SDK.

The methods of **IMediaObject** can be categorized as follows:

### Methods that Handle Format Negotiation

These are the methods that Windows Media Player calls to get information about the data formats supported by the plug-in. These methods include:

- GetInputMaxLatency
- GetInputSizeInfo
- GetInputStreamInfo
- GetInputType
- GetOutputSizeInfo
- GetOutputStreamInfo
- GetOutputType
- GetStreamCount
- SetInputMaxLatency
- SetInputType
- SetOutputType

Several of these methods, such as **GetInputType** and **SetInputType**, use the **DMO_MEDIA_TYPE** structure to describe the format of the data used by a stream. When Windows Media Player calls these methods, it provides a pointer to a **DMO_MEDIA_TYPE** structure. If a method such as **SetInputType** specifies the media type information, the plug-in should copy the **DMO_MEDIA_TYPE** structure to a member variable and inspect its data members to determine the type of data that Windows Media Player will provide in the input buffer. If a method such as **GetInputType** retrieves the media type information, the plug-in should copy the address of the member variable containing the **DMO_MEDIA_TYPE** structure to the pointer provided by
Windows Media Player in the parameter list.

Windows Media Player mainly uses two members of the **DMO_MEDIA_TYPE** structure:

- **majortype**: A globally unique identifier (GUID) that specifies the overall category of the media, such as audio or video.
- **subtype**: A GUID that specifies a more detailed description of the media, such as PCM audio.

These GUIDs can be found in the header named uuids.h, which is included with the DirectX SDK.

Methods such as **GetInputSizeInfo** provide information to Windows Media Player about how much memory is required to allocate the processing buffers. Methods such as **GetStreamCount** and **GetOutputStreamInfo** provide information to Windows Media Player about the number and character of the streams supported by the DSP plug-in.

**GetInputMaxLatency** and **SetInputMaxLatency** are implemented by DMOs in special cases. Windows Media Player DSP plug-ins should return **E_NOTIMPL**.

**Methods that Specify or Retrieve State Information**

These are the methods that Windows Media Player calls to get or set values related to the current state of the plug-in. These methods include:

- **GetInputCurrentType**
- **GetInputStatus**
- **GetOutputCurrentType**

**GetInputCurrentType** and **GetOutputCurrentType** use the **DMO_MEDIA_TYPE** structure to return information to Windows Media Player about the media types previously set for the input and output streams. **GetInputStatus** returns a flag that tells Windows Media Player whether the DSP plug-in can accept input data.

**Methods that Handle Buffering and Processing Data**

These are the methods that Windows Media Player calls to initiate the various processes that the plug-in performs to do the digital signal processing. These methods include:

- **AllocateStreamingResources**
- **Discontinuity**
- **Flush**
- **FreeStreamingResources**
- **Lock**
- **ProcessInput**
- **ProcessOutput**

Windows Media Player calls **AllocateStreamingResources** and **FreeStreamingResources** to provide the DSP plug-in with an opportunity to set up or release any additional buffers the plug-in may require for internal processing.

Windows Media Player DSP plug-ins do not need to use the DMO **Discontinuity** method.

Windows Media Player calls **Flush** to direct the DSP plug-in to flush all internally buffered data. The plug-in should release any references to **IMediaBuffer** interfaces, clear any values that specify the time stamp or sample length for the media buffer, and reinitialize any internal states that depend upon the contents of the
media sample.

Windows Media Player calls **ProcessInput** to pass a pointer to an **IMediaBuffer** interface to the DSP plug-in. This interface provides access to the input buffer allocated by Windows Media Player to supply data to the plug-in. Windows Media Player subsequently calls **ProcessOutput** to pass a pointer to an **IMediaBuffer** interface that provides access to the output buffer allocated by Windows Media Player to receive the processed data from the DSP plug-in.

The **IMediaObject** interface includes a method named **Lock**. This method is designed to acquire or release a lock on the DMO to keep the DMO serialized when performing multiple operations. The version of **IMediaObject::Lock** in the wizard code overrides the ATL implementation of **Lock**. Because the sample code is apartment threaded, the implementation of **Lock** simply returns S_OK. For details about how to create a multi-threaded DMO, refer to the DirectX SDK.

### See Also

- **Required Interfaces**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

About **IWMPPPluginEnable**

The **IWMPPPluginEnable** interface is required for Windows Media Player DSP plug-ins. **IWMPPPluginEnable** contains two methods that store whether Windows Media Player has enabled the DSP plug-in. Windows Media Player calls **IWMPPPluginEnable::SetEnable** when it creates an instance of the DSP plug-in, passing a value of TRUE if it has enabled the plug-in or FALSE otherwise.

DSP plug-ins may remain loaded even when the user chooses to disable them. When disabled, a plug-in must copy data from the input buffer to the output buffer, performing only format conversion processing, if applicable.

Windows Media Player also calls this method before it releases an instance of the DSP plug-in. This is useful to allow clients of the plug-in to check whether the plug-in is currently enabled. For instance, a user interface plug-in might change the appearance of its controls to alert the user that the DSP plug-in is not connected.

### See Also

- **Required Interfaces**

---

© 2000-2003 Microsoft Corporation. All rights reserved.
Registering DSP Plug-ins

Like other COM DLLs, you must register Windows Media Player DSP plug-ins to make them usable. Typically, use the regsvr32.exe utility to manually perform registration tasks. The Windows Media Player Plug-in Wizard creates files that contain registration scripts. These files have an .rgs file name extension. You can also create a setup program that copies your files and registers the DLLs.

You must implement the registration methods on the IWMPMediaPluginRegistrar interface in the implementation blocks of the DllRegisterServer and the DllUnregisterServer functions, which are functions that are exported by COM DLLs. The Windows Media Player registration methods perform the necessary registration housekeeping to allow Windows Media Player to recognize your DLL file as a Windows Media Player plug-in so that it can be presented as an option to the user.

See Also

- DSP Plug-in Developer Overview
- IWMPMediaPluginRegistrar

Building a DSP Plug-in

You can create a Windows Media Player DSP Plug-in by using the Windows Media Player Plug-in Wizard. The Plug-in Wizard is included with the Windows Media Player SDK, which also includes this documentation. If you haven't installed the entire Windows Media Player SDK, you should do so before proceeding.

The Windows Media Player Plug-in Wizard is an add-in for Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET that creates Visual C++ projects and generates sample code for each of the different types of plug-ins supported by Windows Media Player.

The following sections explain what you need to know to create a DSP plug-in using the wizard:

- Getting Started with DSP Plug-ins
- Using the DSP Plug-in Wizard with Visual C++ 6.0
Getting Started with DSP Plug-ins

To set up your development environment for creating DSP plug-ins, you must install the following items:

- Microsoft Visual C++ 6.0 or later
- Windows Media Player 9 Series
- Windows Media Player 9 Series SDK
- Microsoft DirectX 8 SDK
- Windows Media Player Plug-in Wizard

Installing Visual C++

You can install Visual C++ 6.0 or later by itself or as part of Microsoft Visual Studio. For help with installation, refer to the instructions that come with the compiler.

Installing Windows Media Player

Install Windows Media Player 9 Series or any later version. You will need to use it to test your DSP plug-in.

Installing the Windows Media Player SDK

Be sure to install Windows Media Player 9 Series SDK or any later version of the SDK. In addition to this documentation, it includes the Windows Media Player Plug-in Wizard, as well as useful samples.

Installing the Microsoft DirectX 8 SDK

Install version 8.0 or newer of the Microsoft DirectX SDK. You'll need this to get the proper DMO headers and to link with msdmo.lib, which is a required library file for the code that the wizard generates.

Note: The Windows XP SDK includes the DirectX 8 SDK files. If you have already installed the Windows XP SDK, you don't have to install the DirectX 8 SDK.
Installing the Plug-in Wizard

There are separate versions of the Windows Media Player Plug-in Wizard for Microsoft Visual C++ 6.0 and Microsoft Visual C++ .NET.

Once you have installed the Windows Media Player SDK, you can find the Windows Media Player Plug-in Wizard for Visual C++ 6.0 in the following subfolder:

\wizards\wmpplugin

The wizard file is named wmpwiz.awx.

Copy the wizard file to the following subfolder:

\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\IDE

The preceding path assumes you installed Microsoft Visual Studio 6.0 to the default location. If you installed the development environment to a different location, adjust the path accordingly. To quickly locate the correct subfolder, you can use Windows Explorer to search for files with an .awx file name extension.

If you are using Microsoft Visual C++ .NET, the Windows Media Player plug-in wizard is installed and configured automatically for you when you install this SDK. If you install Visual C++ .NET after installing this SDK, you should remove the SDK and reinstall it.

See Also

- Building a DSP Plug-in

Using the DSP Plug-in Wizard with Visual C++ 6.0

Once you have installed the necessary components, creating a sample DSP plug-in is easy. The following steps will guide you:

2. From the File menu, click New…
4. In the Project name box, type a name for your DSP plug-in.
5. In the Location box, provide the location of a folder where the wizard can save the files it generates. You can accept the default, type a new path, or browse to an existing location.
6. Click OK.
7. Click **DSP Plugin**.
8. Click **Audio** or **Video**.
9. Click **Finish**.
10. Click **OK**.

The wizard generates a complete DSP plug-in project.

Before attempting to build the project, be sure to configure your development environment to point to the folders named Include and Lib where you installed the DirectX 8 SDK. Your compiler and linker will need access to some of the files in these folders. In Visual C++ 6.0, you can add these paths on the **Directories** tab of the Options dialog box, which you can access from the **Tools** menu.

**See Also**

- Building a DSP Plug-in

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Using the DSP Plug-in Wizard with Visual C++ .NET**

Once you have installed the necessary components, creating a sample DSP plug-in is easy. The following steps will guide you:

1. Start Microsoft Visual C++ .NET.
2. From the **File** menu, point to **New** and then click **Project**.
3. In **Project Types**, click **Visual C++ Projects** if it isn't already selected.
4. In **Templates**, click **Windows Media Player Plug-in Wizard** to select it.
5. Type a name for your project.
6. Specify a location for your project. This is the folder to which your project files will be copied.
7. Click **OK** to start the wizard.
8. Click **DSP Plug-in**. Click **Next**.
9. Click **Audio** or **Video**.
10. Click **Next**.

The wizard generates a complete DSP plug-in project.

Before attempting to build the project, be sure to configure your development environment to point to the folders named Include and Lib where you installed the DirectX 8 SDK. Your compiler and linker will need access to some of the files in these folders. In Visual C++ .NET, you can add these paths on the **VC++ Directories** page of the **Projects** folder. The **Projects** folder is accessible from the **Options** dialog box, which
you can access from the Tools menu.

See Also

- Building a DSP Plug-in

About the Sample Audio DSP Plug-in

The sample audio DSP plug-in provides a simple processing implementation that scales the amplitude of the audio by a factor provided by the user in the property page. The plug-in accepts values between zero and 1. The default value is 1. A value of 1 has no effect upon the sound; other scale factors are multipliers for the audio samples. For instance, a value of .5 would result in a 6 decibel decrease in volume. A value of zero results in silence.

The sample audio DSP plug-in works with stereo or mono PCM audio.

See Also

- Building a DSP Plug-in

About the Sample Video DSP Plug-in

The sample video DSP plug-in provides a simple processing implementation that scales the color saturation of the video by a factor provided by the user in the property page. The plug-in accepts values between zero and 1. The default value is 1. A value of 1 has no effect upon the video image; other scale factors desaturate the image color. For instance, a value of .5 would result in 50 percent color saturation. A value of zero results in grayscale video.
The sample video DSP plug-in works with the following video formats:

- YV12
- YUY2
- UYVY
- RGB32
- RGB24
- RGB555
- RGB565

See Also

- Building a DSP Plug-in

Implementing Your DSP Code

Once you have built the sample DSP plug-in, you can modify the code to create your own Windows Media Player DSP plug-in. Which methods you change and which you can leave as they are depends upon the following factors:

- Whether your DSP plug-in will process audio or video. Since the sample plug-in processes audio, you will have less code to modify to create an audio DSP plug-in than to create one that processes video. For instance, in addition to changing the code that performs the data processing, creating a video DSP plug-in will require you to alter the implementations of the methods that handle format negotiation between Windows Media Player and the plug-in.
- The number of properties you want to allow the user to change. You will certainly want to change the default property page implementation to suit your needs, and you may need to add additional properties.
- Whether your DSP plug-in needs to allocate any streaming resources. Your plug-in may require additional buffers.
- Whether your audio DSP plug-in needs to continue to output data after Windows Media Player has stopped supplying data in the input buffer.

The following sections use the DSP plug-in sample code generated by the Windows Media Player Plug-in Wizard to illustrate important concepts. You might find it helpful to open Microsoft Visual Studio and generate the sample code first so you can refer to it as you read this section. For details about how to use the Windows Media Player Plug-in Wizard, see Building a DSP Plug-in.
Implementing an Audio DSP Plug-in

To create a Windows Media Player DSP plug-in that processes audio, you'll need to modify the sample code in the function named `DoProcessOutput`. `DoProcessOutput` is called each time Windows Media Player successfully calls `IMediaObject::ProcessOutput`. It is the function that performs the digital signal processing tasks that produce the audible result that the DSP plug-in is intended to produce.

Processing an audio stream is like handling a timed event. `DoProcessOutput` will be called repeatedly and at specific intervals. Each time your code executes, it will need to process a specific number of bytes of data. `DoProcessOutput` contains the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbOutputData</code></td>
<td>This is a <code>BYTE</code> pointer to the buffer where your implementation of <code>DoProcessOutput</code> must copy its processed data.</td>
</tr>
<tr>
<td><code>pbInputData</code></td>
<td>This is a constant <code>BYTE</code> pointer to the buffer that contains the data to be processed.</td>
</tr>
<tr>
<td><code>cbBytesToProcess</code></td>
<td>This is a <code>DWORD</code> value that contains a count of the number of bytes in the input buffer to be processed.</td>
</tr>
</tbody>
</table>

The following sections provide details about how to modify the code generated by the Windows Media Player Plug-in Wizard to create your own audio DSP plug-in:

- Implementing `DoProcessOutput`
- Adding Properties to the Sample Audio DSP Plug-in
Implementing DoProcessOutput

To process audio data, you'll need to perform several steps in DoProcessOutput. The following steps use the plug-in wizard sample code as examples. If you want to create an audio DSP plug-in that processes media content of the same type that the plug-in wizard sample code does, you'll only need to change the actual processing code referred to in step 6. Following are all the steps implemented in DoProcessOutput:

1. If the plug-in is not currently enabled, simply copy the data unchanged into the output buffer. If your plug-in converts the data into a different format, you must also do the conversion processing here.

   // Test whether the plug-in is disabled by the user.
   if (!m_bEnabled)
   {
     // Just copy the data without changing it.
     memcpy(pbOutputData, m_pbInputData, *cbBytesProcessed);

     return S_OK;
   }

2. Retrieve a pointer to the input format structure. You'll need to retrieve member data from this structure, so copy the pointer from m_mtInput.pbformat to a local pointer variable of a type that matches the format structure type. The following example stores a pointer to a WAVEFORMATEX input format structure:

   WAVEFORMATEX *pWave = (WAVEFORMATEX*) m_mtInput.pbFormat;

3. Calculate the number of samples to process. The sample code that the plug-in wizard generates performs this step by dividing the number of bytes to process by the nBlockAlign member of the WAVEFORMATEX input format structure, and then multiplying the result by the number of channels, which was stored in the nChannels member. The following example is from the plug-in wizard sample code:

   DWORD dwSamplesToProcess = (cbBytesProcessed / pWave->nBlockAlign) * pWave->nChannels
4. Determine the bit depth of the audio. The plug-in wizard sample code determines 8-bit or 16-bit audio by inspecting the `wBitsPerSample` member of the `WAVEFORMATEX` structure. It then uses that value in a switch statement to provide separate processing routines for each bit depth. You may need to use a different technique when dealing with other format types and bit depths.

5. Create a loop to step through the audio samples in the input buffer.

6. Retrieve a sample from the input buffer. You do this by dereferencing the input data pointer and storing the result in a variable of type `int`. For 16-bit audio, you must recast the `BYTE` pointer to a `short` pointer to handle the greater audio sample precision. Once you have the value, you can immediately increment the `pbInputData` pointer so that it points to the next sample. The following examples demonstrate this:

   ```c
   // For 8-bit audio.
   int i = *pbInputData++;
   
   -or-
   
   // For 16-bit audio.
   // Recast the pointer.
   short *pwInputData = (short *) pbInputData;
   
   // Enter the loop and then get the input sample.
   int i = *pwInputData++;
   ```

7. Perform the processing. This is where you apply the algorithms that change the sample somehow. What you do here is up to you.

8. Write the processed data to the output buffer. Immediately increment the pointer to the output buffer, as in the following example:

   ```c
   *pwOutputData++ = i;
   ```

9. Repeat the loop until all the samples have been processed.

10. Return an appropriate HRESULT.

See Also

- [Implementing an Audio DSP Plug-in](#)

Adding Properties to the Sample Audio DSP Plug-in

The audio DSP sample code that the Windows Media Player Plug-in Wizard generates uses a single property that represents the scale factor for the audio volume. Your plug-in may require more than one property. You can easily add properties to your DSP plug-in in Visual Studio using the following steps:
1. Define the methods in the interface definition code in the project's main header file. For example, to add a property called "color", you would create two accessor methods using the following syntax:

```cpp
virtual HRESULT STDMETHODCALLTYPE get_color(COLORREF *pColor) = 0;
virtual HRESULT STDMETHODCALLTYPE put_color(COLORREF newColor) = 0;
```

2. Add the method declarations to the main class declaration in the header file:

```cpp
STDMETHOD(get_color)(COLORREF *pColor);
STDMETHOD(put_color)(COLORREF newColor);
```

3. Add the method implementations to the project's main CPP file:

```cpp
STDMETHODIMP CYourProject::get_color(COLORREF *pColor)
{
    if ( NULL == pColor )
    {
        return E_POINTER;
    }
    *pColor = m_Color;
    return S_OK;
}

STDMETHODIMP CYourProject::put_color(COLORREF newColor)
{
    m_Color = newColor;
    return S_OK;
}
```

Finally, to make the properties accessible to the user, you'll want to make changes to the property page implementation.

See Also

- Implementing an Audio DSP Plug-in

Implementing the Property Page for a DSP Plug-in

Windows Media Player can display a property page for each DSP plug-in to enable users to set values that change the behavior of the plug-in. Users can access the property page from the Plug-ins tab of the Options dialog box by clicking the name of the DSP plug-in to select it and then clicking Properties.
The Windows Media Player Plug-in Wizard sample code provides a default implementation of a property page that includes a single edit box that receives from the user a value representing a scale factor for the audio volume. When the user clicks **Apply**, the property page passes this value to the plug-in object so the plug-in can change the value it uses to scale the volume during processing.

**About the Property Page Object**

The property page object is a COM object that implements the `IPropertyPage` interface. The sample code generated by the plug-in wizard uses the ATL implementation of `IPropertyPage`, which is documented in the Visual C++ documentation on MSDN. Your code at least should provide override implementations for `IPropertyPage::OnInitDialog`, which handles the event that occurs when the property page opens, and `IPropertyPage::Apply`, which handles the event that occurs when the user clicks **Apply**. The plug-in wizard generates sample code for each of these event handlers. The sample implementation of `OnInitDialog` retrieves a value from the registry in order to display the current scale factor setting. The sample implementation of `Apply` validates the user input by testing to ensure that it falls within a specified range, then persists the value to the registry, and then passes the value (if valid) to the DSP plug-in property put method, named `put_scale`.

Additionally, you will need to add code to handle the event that occurs when the user changes a value in a control you provide in the property page. For example, the plug-in wizard implements a function named `OnChangeScale`, which simply enables the **Apply** button when the user changes the text in the edit box on the property page by calling the `SetDirty` method with an argument value of TRUE.

**About the Property Page Dialog Resource**

The user interface for the property page is stored as a dialog resource. You can easily view and edit the property page dialog in Visual Studio by selecting the **ResourceView** tab in the Project Workspace window, then opening the **Dialog** folder, and then double-clicking the property page resource name. The dialog resource editor in Visual Studio provides you with the tools you need to add controls to the property page dialog and to create event handlers that are mapped to the appropriate Windows messages. For details about how to use the resource editor in Visual Studio, refer to Visual Studio Help.

**About ISpecifyPropertyPages**

If a DSP plug-in provides a property page, the plug-in must implement the `ISpecifyPropertyPages` interface. This interface contains only one method: `ISpecifyPropertyPages::GetPages`. This is the method that associates the property page with the DSP plug-in. Windows Media Player calls this method when the user invokes the property page, passing a parameter of type CAUUID, which is a counted array of GUIDs. The sample plug-in implementation of `GetPages` fills this structure with a single GUID that is the class id of the plug-in property page object. Windows Media Player then uses the class id to create the property page object.

You might notice that the sample implementation of `GetPages` uses `CoTaskMemAlloc` to allocate memory for the GUID structure. It is the responsibility of the caller, in this case Windows Media Player, to use `CoTaskMemFree` to release the memory. For details about the CAUUID structure, see the Platform SDK documentation.

See Also

- [Implementing an Audio DSP Plug-in](#)
Changing the Sample Audio DSP Plug-in Property

You will probably want to change the property that the Windows Media Plug-in Wizard creates by default. The following list details the items that might require changing:

- **The dialog resource.** Click the ResourceView tab in the Project Workspace window. Expand the folder list to open the Dialog folder. Double-click the dialog resource to open the resource editor. You can make changes to the property page dialog to fulfill your needs. For instance, you could change the text in the label or replace the edit control with a checkbox.

- **The property page object code.** The default implementation uses a variable of type double to store the scale factor. You might require a different type of data. This would also require you to change the code that persists the data to the registry and reads the data from the registry (including the code that reads from the registry in `CProjectName::FinalConstruct`).

- **The member variable that stores the property value.** This variable is named `m_fScaleFactor` and is declared as type double. You may want to change the name and type of this variable throughout the project.

- **The property get and property put methods.** You might want to change the names, parameters, and implementations of these methods. Don't forget to also reflect those changes elsewhere in the project. For instance, the property page `Apply` method calls `CProjectName::put_scale`.

See Also

- [Implementing an Audio DSP Plug-in](#)
The sample code is written to always force the client to process all output before the plug-in will accept more input.

The default implementation of `IMediaObject::Discontinuity` simply returns S_OK.

See Also

- Implementing an Audio DSP Plug-in

About Allocating Streaming Resources

The sample DSP plug-in generated by the Windows Media Player Plug-in Wizard does not require any additional streaming buffers. However, you might want to allocate memory resources for your DSP plug-in. For example, a plug-in that produces an echo effect would require a secondary buffer to create the necessary time delay.

The `IMediaObject` interface contains two methods to handle this situation. Windows Media Player calls `IMediaObject::AllocateStreamingResources` to give you an opportunity to create any buffers you require. Windows Media Player later calls `IMediaObject::FreeStreamingResources` to allow you to free any memory you allocated previously. The sample DSP plug-in implementation also calls `FreeStreamingResources` from `CProjectName::FinalRelease` to ensure that all resources are freed before the plug-in object is destroyed.

See Also

- Implementing an Audio DSP Plug-in

Implementing a Video DSP Plug-in
Computer video display adapters support a set of video formats. Digital video codecs also support a set of video formats. When attempting to play a particular video file, Windows Media Player must choose a format to use for rendering. The Player attempts to find the best match between the formats supported by the video codec and the formats supported by the video display adapter—that is, the one that yields the highest quality.

To create a Windows Media Player DSP plug-in that processes video, you'll first need to decide which video formats you'd like your plug-in to process. The sample video DSP plug-in works with the following video formats:

- YV12
- YUY2
- UYVY
- RGB32
- RGB24
- RGB555
- RGB565

Which formats you choose to process is up to you. You can remove formats from the sample plug-in so that they aren't supported any longer and you can add code to process additional formats.

The following sections provide additional information you should know before creating your own video DSP plug-in for Windows Media Player:

- Video Format Negotiation in the Sample Video DSP Plug-in
- DoProcessOutput in the Sample Video DSP Plug-in
- Processing the Video

See Also

- Implementing Your DSP Code
plug-in can respond with the media type that corresponds to the index value. The plug-in is not required to support more than one format, but using this mechanism it can communicate to the Player the details of each format that it supports. It does this by filling in a DMO_MEDIA_TYPE structure using a pointer provided by the Player. When the Player provides an index number that exceeds the highest supported media type index value for the plug-in, the plug-in returns E_NO_MORE_ITEMS.

The sample video plug-in generated by the Windows Media Player Plug-in Wizard stores the list of supported video formats as an array of GUIDs. The following code is from the main header file:

```cpp
static const GUID* k_guidValidSubtypes[] = {
    &MEDIASUBTYPE_YV12,
    &MEDIASUBTYPE_YUY2,
    &MEDIASUBTYPE_UYVY,
    &MEDIASUBTYPE_RGB32,
    &MEDIASUBTYPE_RGB24,
    &MEDIASUBTYPE_RGB555,
    &MEDIASUBTYPE_RGB565
};
```

These formats are added to the array in order of preference for the plug-in. When the Player calls `GetInputType`, the plug-in returns the subtype that corresponds to the `dwTypeIndex` value requested by the Player. The following code from the sample implementation of `GetInputType` demonstrates this:

```cpp
::ZeroMemory( pmt, sizeof( DMO_MEDIA_TYPE ) );
pmt->majortype = MEDIATYPE_Video;
pmt->subtype = *k_guidValidSubtypes[dwTypeIndex];
```

The Player can call `GetInputType` and `GetOutputType` in any order, so the plug-in code must anticipate this. The sample implementation of `GetOutputType` tests whether the input type has already been defined. If it has, the plug-in only responds that it supports that type. Otherwise, the plug-in returns the type that corresponds to the supplied index, as the following code demonstrates:

```cpp
// If input type has been defined, then use that as output type.
if (GUID_NULL != m_mtInput.majortype)
{
    hr = ::MoCopyMediaType( pmt, &m_mtInput );
}
else // otherwise use default for this plug-in
{
    ::ZeroMemory( pmt, sizeof( DMO_MEDIA_TYPE ) );
    pmt->majortype = MEDIATYPE_Video;
    pmt->subtype = *k_guidValidSubtypes[dwTypeIndex];
}
```

### Setting the Input and Output Types

When Windows Media Player is ready to set the media type for the plug-in, it calls `IMediaObject::SetInputType` and `IMediaObject::SetOutputType`, passing to each function a pointer to a DMO_MEDIA_TYPE structure that represents the requested media type. Note that there is no guarantee that the Player will call format negotiation methods in any particular order, so plug-in code must handle any case. For instance, if the Player calls `SetOutputType` before calling `SetInputType`, it is a valid course of action for the plug-in to reject the proposed output media type. The following code from the sample implementation of `SetOutputType` demonstrates this:

```cpp
if( GUID_NULL != m_mtInput.majortype )
{
    // validate that the output media type matches our requirements
```
The sample plug-in implementations of `SetInputType` and `SetOutputType` both call the custom function named `ValidateMediaType`. This plug-in function performs a series of tests on the proposed media type designed to ensure that the media type is well-formed and supported by the plug-in. `ValidateMediaType` performs the following tests:

- Verifies that the `majortype` and `formattype` members contain the correct values.
- Verifies that the `subtype` member matches one of the supported formats.
- Verifies that the information in the `BITMAPINFOHEADER` and `VIDEOINFOHEADER` or `VIDEOINFOHEADER2` structures contain valid values.
- Tests whether the input and output media types match because the plug-in does not convert formats from input to output.

If the proposed media type passes the validation tests, it is stored in a member variable: `m_mtInput` for the input media type, `m_mtOutput` for the output media type.

See Also

- [Implementing a Video DSP Plug-in](#)

DoProcessOutput in the Sample Video DSP Plug-in

Because a video DSP plug-in typically supports several video formats, it is convenient to separate the processing implementation code into a separate function for each format. This means that the implementation of `IMediaObject::DoProcessOutput` for video DSP plug-ins is relatively simple. The implementation in the sample plug-in first tests whether the user has enabled the plug-in. If the plug-in is disabled, the code copies the data provided in the input buffer to the output buffer without changing it, as the following code demonstrates:

```c
// Test whether the plug-in has been disabled by the user.
if (!m_bEnabled)
{
    // Just copy the data without changing it. You should // also do any neccessary format conversion here.
    memcpy(pbOutputData, pbInputData, m_dwBufferSize);
    *cbBytesProcessed = m_dwBufferSize;
}
If the plug-in is enabled, the code simply performs a series of checks on the input media type *subtype* member to determine the current video format. When a match is found, the code calls the appropriate processing function.

**See Also**

- [Implementing a Video DSP Plug-in](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Processing the Video**

The details of processing video vary for each format; it is beyond the scope of this documentation to provide these details. In a general sense, the goal of the plug-in is to change the color data in the input buffer and then copy the data to the output buffer.

The sample plug-in processes two types of video formats: YUV and RGB. For YUV video, the red and blue color information is encoded in the U and V values and the luminance level is represented by the Y value; the green value is encoded and can be recovered by using an algorithm. The sample plug-in simply changes the U and V values to affect the color level. Each U or V byte has a value between zero and 255. The plug-in first adjusts each value to be represented by a range from -128 to 127, and then scales the value by the supplied scale factor. Finally, the code adjusts the value again for the original zero-to-255 range and copies the data to the output buffer. The following example code processes UYVY video. In this format, every other byte is a U or Y value.

```c
while( dwHeight-- )
{
    DWORD x = dwWidth;
    while( x-- )
    {
        // Scale the U and V bytes to 128.
        // Just copy the Y bytes.
        if( x%2 )
        {
            pbTarget[x] = pbSource[x];
        }
        else
        {
            long temp = (long)((pbSource[x] - 128) * m_fScaleFactor);
            // Truncate if exceeded full scale.
            if (temp > 127)
            {
                long temp2 = temp & 0xFF;
                long temp3 = temp2 + 128;
                pbTarget[x] = (BYTE)temp3;
            }
            else
            {
                long temp2 = temp & 0xFF;
                pbTarget[x] = (BYTE)temp2;
            }
        }
    }
}
```
For RGB video, the color and luminance information is encoded as separate red, green, and blue values. The sample plug-in computes the average of the three values to determine the value for gray, and then adjusts each color value by using the supplied scale factor. Once again, the values must be normalized for the -128 to 127 range before scaling. The following code from \textbf{Process32Bit} shows the process for RGB32:

```c
while( dwHeight-- )
{
    RGBQUAD* pPixelIn = (RGBQUAD*)pbSource;
    RGBQUAD* pPixelOut = (RGBQUAD*)pbTarget;

    for( DWORD x = 0; x < dwWidth; x++ )
    {
        // Get the color bytes.
        long lBlue = (long) pPixelIn[x].rgbBlue;
        long lGreen = (long) pPixelIn[x].rgbGreen;
        long lRed = (long) pPixelIn[x].rgbRed;

        // Compute the average for gray.
        long lAverage = ( lBlue + lGreen + lRed ) / 3;

        // Scale the colors to the average.
        pPixelOut[x].rgbBlue = (BYTE)( ( lBlue - lAverage ) * m_fScaleFactor + lAverage );
        pPixelOut[x].rgbGreen = (BYTE)( ( lGreen - lAverage ) * m_fScaleFactor + lAverage );
        pPixelOut[x].rgbRed = (BYTE)( ( lRed - lAverage ) * m_fScaleFactor + lAverage );
        pPixelOut[x].rgbReserved = pPixelIn[x].rgbReserved;
    }

    // Move the pointers to the next row.
    pbSource += lStrideIn;
    pbTarget += lStrideOut;
}
```

For more information about processing video, download the DirectX SDK from the \textbf{Microsoft Web site}. For more information about video formats, see the \textbf{FourCC Web site}.

\textbf{See Also}

- \textbf{Implementing a Video DSP Plug-in}
DSP Plug-ins Programming Guide

This section describes how to modify the code generated by the Windows Media Player Plug-in Wizard to create your own Windows Media Player DSP plug-in. This section contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Echo Sample</td>
<td>Explains how to create a DSP plug-in that adds an echo effect to the audio.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player DSP Plug-ins

The Echo Sample

The Windows Media Player Plug-in Wizard can create a DSP plug-in project for Microsoft Visual C++ 6.0. The default code generated by the wizard allows the user to provide a scale factor between 0 and 1, which is used by the program as a multiplier for the audio samples. This is a very simple implementation that you can study to understand how Windows Media Player interacts with DSP plug-ins. The information in the section named About DSP Plug-ins can help you to understand the default implementation.

The sample described in this section is a bit more complex. This sample allows the user to specify a delay time, in milliseconds, and an effect level. The code uses these values to generate an echo effect when playing files that contain pulse code modulation (PCM) audio. Many of the file types that Windows Media Player renders use PCM audio.

This guide is divided into the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo Sample Overview</td>
<td>Describes the general requirements and specifications</td>
</tr>
</tbody>
</table>
Echo Sample Overview

This guide builds a Windows Media Player DSP plug-in that creates an echo effect in PCM audio during playback. The goals for the plug-in are as follows:

- The plug-in processes 8-bit or 16-bit PCM audio only.
- It supports a delay time between 10 milliseconds (ms) and 2000 ms (2 seconds). This represents a practical range for most applications.
- It supports mixing of the original signal with the delay signal.
- It provides a property page implementation that allows the user to provide a value for the delay time and a value for the percentage of delay signal relative to the overall audio signal level.
- The code is created by modifying the Windows Media Player Plug-in Wizard audio DSP plug-in sample.

The Echo sample is not included with the Windows Media Player SDK; it is a sample that you create. To create the Echo sample, you must start with the default project from the Windows Media Player Plug-in Wizard. You can name the project whatever you like; this documentation assumes the project is named Echo. For details about using the wizard, see Building a DSP Plug-in.

The following section provides an overview of how the sample creates an echo effect:
How the Echo Sample Works

The code creates the echo effect by allocating a buffer large enough to contain exactly the amount of audio data that can be rendered in the time frame specified by the delay time value. The size of the buffer is calculated, in bytes, by the following formula:

\[
\text{buffer size} = \text{delay time} \times \text{sample rate} / 1000 \times \text{block alignment}
\]

The delay time is in milliseconds. The sample rate and block alignment values are given in a WAVEFORMATEX structure. The sample rate is in samples per second; dividing by 1000 yields samples per millisecond. The block alignment is equal to the product of the number of channels (1 for mono, 2 for stereo) and the number of bits per sample (8 or 16) divided by 8 (bits per byte).

In addition to the pointer variable that points to the head of the delay buffer, the code creates a movable pointer that steps through the data in the buffer in synchronization with the processing loop in the DoProcessOutput function. When the movable pointer reaches the end of the delay buffer, it moves back to the head of the buffer. A buffer used in this manner is called a circular buffer.

Once the delay buffer exists, and Windows Media Player has allocated an input buffer to supply audio data and an output buffer to receive processed audio data, the echo processing proceeds like this:

1. Enter a loop that allows processing of each audio sample in the input buffer.
2. Retrieve a sample from the input buffer. Then, move the input buffer pointer forward to the next sample to prepare for the next loop iteration.
3. Retrieve a sample from the delay buffer.
4. Copy the sample from the input buffer to the same location in the delay buffer from which the last delay sample was retrieved.
5. Move the delay buffer pointer forward to the next sample. If the pointer moves past the end of the buffer, move it to the head of the buffer.
6. Combine the sample from the input buffer with the sample from the delay buffer.
7. Copy the result to the output buffer. Then, move the output buffer pointer forward to the next unit to prepare for the next loop iteration.
8. Repeat until all the samples are processed.

When an input sample retrieved in step 2 is copied to the delay buffer in step 4, it remains there until the
movable pointer steps through each sample in the delay buffer and finally returns to the same position. Because the size of the delay buffer is designed to correspond to the delay time, the elapsed time between the sample being copied to the delay buffer and the sample being retrieved once again equals the specified delay (plus any latency introduced by the actual processing).

When a stream starts, no delay data is generated until the delay time has elapsed. Therefore, it is important that the delay buffer initially contains silence. If the delay buffer contains random data, the user will hear white noise until the plug-in generates enough delay data to overwrite the entire delay buffer.

See Also

- Echo Sample Overview

Echo Sample Properties

The Echo sample exposes two properties: the delay time and the effect level (wet mix). The value for the dry signal level (dry mix) is always derived from the wet mix value. You need to modify existing code and add some new code to make these properties accessible.

The following sections explain how to modify the properties code:

- How Properties Work
- Variables to Store Properties
- Modifying the Scale Property
- Adding the Wet Mix Property

See Also

- The Echo Sample
How Properties Work

Property values are stored in member variables.

Properties are made accessible by method pairs. One method provides the implementation to specify the property value; its name starts with put_. The other method provides the implementation to retrieve the property value; its name starts with get_. The interface definition and property method prototypes are in Echo.h. They are implemented in Echo.cpp.

The next three sections will show you how to modify the existing property methods to suit your needs and how to add the methods for an additional property.

- Variables to Store Properties
- Modifying the Scale Property
- Adding the Wet Mix Property

See Also

- Echo Sample Properties

Variables to Store Properties

First, you will need a variable to store the delay time. The default sample created by the Windows Media Player Plug-in Wizard provides a variable named m_fScaleFactor to store the scaling multiplier it uses for processing. This sample no longer needs this variable, so you can change its name and type to store the delay time value.

1. Replace each instance of m_fScaleFactor in Echo.h and Echo.cpp with m_dwDelayTime.
2. Change the data type for m_fScaleFactor (now m_dwDelayTime) from double to DWORD in Echo.h.
3. In the constructor for CEcho, change the default delay time value to 1000.

    m_dwDelayTime = 1000;   // Default to a delay time of 1000 ms.

Next, declare two new member variables to store the percentage of effect signal and the percentage of source signal to be mixed in the final output buffer. The term "wet" refers to the effect, and the term "dry" refers to the source signal. Add the following declarations to Echo.h:

```cpp

```
double m_fWetMix;    // percentage of effect
double m_fDryMix;    // percentage of dry signal

These values are stored as decimal representations of percentages so they can easily be used as scale factors. For instance, a mixture of 50 percent effect and 50 percent source signal would be represented as a value of 0.50 for each variable. The sum of the values for m_fWetMix and m_fDryMix must not be more than 1.0 (100 percent). Eventually, these values will be accessible as properties.

Add the following code to the CEcho constructor to set the default values to 50 percent each:

m_fWetMix = 0.50;  // default to 50 percent wet
m_fDryMix = 0.50;  // default to 50 percent dry

See Also

- Echo Sample Properties

Modifying the Scale Property

The default wizard implementation exposes the scale property. You can change the existing implementation to expose the delay time property instead.

First, use the following example to change the function prototypes for get_scale and put_scale in Echo.h. Change the name of the methods and the data types for the parameters:

```c
// IEcho methods
STDMETHOD(get_delay)(DWORD *pVal);
STDMETHOD(put_delay)(DWORD newVal);
```

Next, change the implementations of the get_scale and put_scale methods in Echo.cpp. Make the code match the following examples:

```c
// Formerly get_scale
STDMETHODIMP CEcho::get_delay(DWORD *pVal)
{
    if ( NULL == pVal )
    {
        return E_POINTER;
    }
    *pVal = m_dwDelayTime;
    return S_OK;
```
The preceding example code changes the method names and the parameter data types. The member variable name should have been changed previously. Remember to change the comments that introduce each method as well.

Now, change the interface definition. The following code replaces the code in the IEcho interface declaration in Echo.h:

```cpp
virtual HRESULT STDMETHODCALLTYPE get_delay(DWORD *pVal) = 0;
virtual HRESULT STDMETHODCALLTYPE put_delay(DWORD newVal) = 0;
```

See Also

- [Echo Sample Properties](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

## Adding the Wet Mix Property

You must add the code to provide the additional property for the effect level.

The section [Adding Properties to the Sample Audio DSP Plug-in](#) provides details about how to add a new property using Visual C++. This section shows you how to add the code manually. This entails adding code in the same three places where you modified the code for the delay time property.

Add the prototypes for the get_wetmix and put_wetmix methods immediately following the other property method prototypes in Echo.h. Use the following syntax:

```cpp
STDMETHOD(get_wetmix)(double *pVal);
STDMETHOD(put_wetmix)(double newVal);
```

Now, add the implementation for each method immediately following the other property implementations in Echo.cpp. The following example shows the code for both methods:

```cpp
// Property get to retrieve the wet mix value by using the public interface.
STDMETHODIMP CEcho::get_wetmix(double *pVal)
```
if ( NULL == pVal )
{
    return E_POINTER;
}
*pVal = m_fWetMix;
return S_OK;

// Property put to store the wet mix value by using the public interface.
STDMETHODIMP CEcho::put_wetmix(double newVal)
{
    m_fWetMix = newVal;

    // Calculate m_fDryMix
    m_fDryMix = 1.0 - m_fWetMix;

    return S_OK;
}

Notice that the implementation of put_wetmix includes the code to calculate the correct value for m_fDryMix. Each time a new value is specified for m_fWetMix, this calculation is required.

Add the following code in the interface definition just after the code for the delay methods in Echo.h:

virtual HRESULT STDMETHODCALLTYPE get_wetmix(double *pVal) = 0;
virtual HRESULT STDMETHODCALLTYPE put_wetmix(double newVal) = 0;

See Also

- Echo Sample Properties

Modifying the Echo Sample Property Page

The default property page implementation provided by the Windows Media Player Plug-in Wizard DSP plug-in sample contains a single edit box control that receives the scale factor from the user. You need to modify the property page to receive the two property values used by the Echo sample.

For an overview of DSP plug-in property pages, see Implementing an Audio DSP Plug-in.

The following sections step you through the process of modifying the sample property page:
Modifying the Echo Dialog Resource

You need to change the dialog resource that is the user interface for the property page object. You can first change the existing edit box and label to be useful for the delay time property and then add a second edit box and label for the wet mix property.

To edit the dialog resource in Visual C++:

1. Click the ResourceView tab in the Project Workspace.
2. Expand the resources tree by opening the top level folder.
3. Open the Dialog folder.
4. Double-click the dialog resource name, IDD_ECHOPROPPAGE. The resource editor appears in the right pane.

Changing the Existing Resources

To change the existing property page resources for the delay time property:

1. First, change the text in the existing static text control. Right-click the control and then choose Properties. In the Caption field, type the new caption:

   Delay time (0 to 2000):

2. Close the Text Properties dialog box.
3. Now, change the name of the edit box control. To do this, right-click the control and then choose Properties. In the ID field, type a new name for the control:

   IDC_DELAYTIME

5. Save the resource.
6. Answer Yes if prompted to reload the file resource.h.
7. Click the FileView tab in the Project Workspace. Open resource.h
8. Locate the #define for the scale factor edit box resource (IDC_SCALEFACTOR) and delete it. It should have the same id number as IDC_DELAYTIME.

Adding the New Resources

To add the new property page resources for the wet mix property:

1. Click the ResourceView tab in the Project Workspace to select it.
2. Double-click the name of the property page dialog box, IDD_ECHOPROPPAGE. The resource editor appears in the right pane.
3. Use the toolbox to add a static text control and an edit box to the property page.
4. Right-click the static text control and choose Properties.
5. Type a new name for the static text control in the ID field:

   IDC_MIXLABEL

6. Type a caption for the label:

   Effect level (%):

7. Close the Text Properties dialog box.
8. Right-click the edit box and choose Properties.
9. Type a new name for the edit box in the ID field:

   IDC_WETMIX


When you save the project, you may be prompted to reload resource.h. Click Yes if this happens. The dialog box resource editor should add the resource names and id numbers to resource.h for the items you added. If for some reason this doesn't happen, you must open resource.h and type new entries for the label and edit box control, and assign each a unique id number.

Modifying and Adding the String Resources

The plug-in wizard sample code specifies a string resource named IDS_SCALERANGEERROR that contains a message to display when the user input is out of range. You can modify this resource to suit your needs for the delay time value by following these steps in Visual C++:

1. Click the ResourceView tab.
2. Open the String Table folder.
3. Double-click the String Table icon to open the resource editor.
4. Double-click the name of the resource you want to edit, in this case, IDS_SCALERANGEERROR. The String Properties dialog box appears.
5. Change the name in the ID field to IDS_DELAYRANGEERROR.
6. Change the text in the Caption field:

   You must enter a delay time between 0 and 2000 milliseconds.

7. Close the String Properties dialog box.
Next, add a new string resource for the wet mix property error message.

8. Double-click the empty line at the bottom of the resource editor.
9. Change the name in the ID field to IDS_MIXRANGEERROR.
10. Add the following text to the Caption field:

    You must enter an effect level between 0 and 100 percent.


There are two other values you will want to change in the String Table. IDS_FRIENDLYNAME is the name that appears in the Windows Media Player user interface to identify the plug-in. IDS_DESCRIPTION lets you tell the user about your plug-in. Both of these strings are passed as parameters to the IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin function, which is called in the DllRegisterServer method in Echodll.cpp.

See Also

- Modifying the Echo Sample Property Page
Adding the Wet Mix Event Handler

You can easily add the event handler for the EN_CHANGE event that is attached to the IDC_WETMIX edit box control. From the dialog resource editor:

1. Right-click the IDC_WETMIX edit box and choose Events. The New Windows Message and Event Handlers dialog box appears.
2. In the Class or object to handle box, click the name of the edit box resource, IDC_WETMIX.
3. In the New Windows messages/events box, click EN_CHANGE to select it.
4. Click Add Handler. The Add Member Function dialog box appears.
5. In the Member function name box, type the name OnChangeWetmix.
6. Click OK to close the Add Member Function dialog box.
7. Click OK to return to the dialog box resource editor.

Visual C++ automatically adds the code for the message map and for the event handler function to EchoPropPage.h. The code it inserts provides a TODO comment where you can add the implementation in the header for the function. This is a slightly different style than the Windows Media Player Plug-in Wizard sample code uses, but is acceptable.

Whether you want to write your implementation in the header file or move it to EchoPropPage.cpp is up to you. In either case, the implementation needs only a single additional line of code to enable Apply in the property page dialog. Insert this line of code before the line that returns from the function:

SetDirty(TRUE); // Enable Apply.

See Also

- Modifying the Echo Sample Property Page

How the Echo Sample Persists Data

When Windows Media Player enables a DSP plug-in, it may create and destroy many instances of the plug-in object during the course of a session. The plug-in needs a way to persist its property values between instances. The sample code generated by the Windows Media Player Plug-in Wizard stores these values in the registry and retrieves them when the property page is invoked or when a new instance of the plug-in is created.

The default sample code in Echo.h includes two constants that store the default registry path and the scale factor name string. You should keep the variable that specifies the path, but delete the line that specifies the scale factor registry name. Then, add the following code to define constants for the delay time and wet mix property
names in the registry. The finished section should appear as follows:

```cpp
// registry location for preferences
const TCHAR kszPrefsRegKey[] = _T("Software\Echo\DSP Plugin");
const TCHAR kszPrefsDelayTime[] = _T("DelayTime");
const TCHAR kszPrefsWetmix[] = _T("Wetmix");
```

You will use these constants when you modify the property page methods.

**See Also**

- Modifying the Echo Sample Property Page
The dialog box displays the effects level value to the user as an integer, but the plug-in stores the value as a floating point number. Therefore, the code converts the floating point value to a DWORD value.

**Retrieving the Current Values from the Registry**

If the property page cannot retrieve the current values from the plug-in, it must attempt to read them from the registry. The following code reads each property value:

```cpp
else // Otherwise, read values from registry
{
    CRegKey key;
    LONG lResult;

    lResult = key.Open(HKEY_CURRENT_USER, kszPrefsRegKey, KEY_READ);
    if (ERROR_SUCCESS == lResult)
    {
        DWORD dwValue = 0;

        // Read the delay time.
        lResult = key.QueryValue(dwValue, kszPrefsDelayTime);
        if (ERROR_SUCCESS == lResult)
        {
            dwDelayTime = dwValue;
        }

        // Read the wet mix value.
        lResult = key.QueryValue(dwValue, kszPrefsWetmix);
        if (ERROR_SUCCESS == lResult)
        {
            dwWetmix = dwValue;
        }
    }
}
```

Notice the use of the registry constants you created previously.

**Displaying the Values to the User**

Finally, the property page must display the values in the correct edit box controls. The following example code demonstrates this:

```cpp
TCHAR szStr[MAXSTRING];

// Display the delay time.
_stprintf(szStr, _T("%u"), dwDelayTime);
SetDlgItemText(IDC_DELAYTIME, szStr);

// Display the effect level.
_stprintf(szStr, _T("%u"), dwWetmix);
SetDlgItemText(IDC_WETMIX, szStr);
```

**See Also**

- [Modifying the Echo Sample Property Page](#)

**Previous**

**Next**
Implementing CEchoPropPage::Apply

The CEchoPropPage::Apply method is implemented in EchoPropPage.cpp. It executes when the user clicks Apply in the property page dialog box in Windows Media Player. The plug-in wizard sample code provides an implementation to handle a single property. You can modify this code for one of the Echo sample properties, and then add code to store the other property value.

Declaring the Apply Method Variables

First, you must remove the declaration of fScaleFactor. Then, add variable declarations that you will need. The following example shows the completed variable declarations:

```c++
TCHAR szStr[MAXSTRING] = { 0 };  
DWORD dwDelayTime = 1000;  // Initialize the delay time.  
DWORD dwWetmix = 50;       // Initialize a DWORD for effect level.  
double fWetmix = 0.50;      // Initialize a double for effect level.
```

Retrieving the Values from the Property Page

You must implement code to retrieve and validate the user input. The following code example retrieves the delay time value from the IDC_DELAYTIME edit box, and then verifies that the value is within a specified range:

```c++
// Get the delay time value from the dialog box.
GetDlgItemText(IDC_DELAYTIME, szStr, sizeof(szStr) / sizeof(szStr[0]));

dwDelayTime = atoi(szStr);

// Make sure delay time is valid.
if ((dwDelayTime < 10) || (dwDelayTime > 2000))
{
    if (::LoadString(_Module.GetResourceInstance(), IDS_DELAYRANGEERROR, szStr, sizeof(szS
    
    MessageBox(szStr);
    
    return E_FAIL;
}
```

If the user input is not in the specified range, the code displays a message box. Notice the use of the string resource you created earlier for the error message.

The following example retrieves the effect level from the IDC_WETMIX edit box and then verifies that the value is within a specified range:

```c++
// Get the effects level value from the dialog box.
GetDlgItemText(IDC_WETMIX, szStr, sizeof(szStr) / sizeof(szStr[0]));
```
dwWetmix = atoi(szStr);

// Make sure wet mix value is valid.
if ((dwWetmix < 0) || (dwWetmix > 100))
{
    if (::LoadString(_Module.GetResourceInstance(), IDS_MIXRANGEERROR, szStr, sizeof(szStr)
        MessageBox(szStr);
    }

return E_FAIL;
}

Storing the Property Values in the Registry

Next, your code must persist the new property values to the registry. The following code stores both property values:

// update the registry
CRegKey key;
LONG lResult;

// Write the delay time value to the registry.
lResult = key.Create(HKEY_CURRENT_USER, kszPrefsRegKey);
if (ERROR_SUCCESS == lResult)
{
    lResult = key.SetValue( dwDelayTime , kszPrefsDelayTime );
}

// Write the wet mix value to the registry.
lResult = key.Create(HKEY_CURRENT_USER, kszPrefsRegKey);
if (ERROR_SUCCESS == lResult)
{
    lResult = key.SetValue( dwWetmix , kszPrefsWetmix );
}

Updating the Echo Plug-in Property Values

The Apply method must inform the Echo plug-in that the property values have changed. The following code calls the property put method for each property using the interface pointer retrieved in CEchoPropPage::SetObjects:

// update the plug-in
if (m_spEcho)
{
    m_spEcho->put_delay(dwDelayTime);

    // Convert the wet mix value from DWORD to double.
    fWetmix = (double)dwWetmix / 100;
    m_spEcho->put_wetmix(fWetmix);
}

Notice that the wet mix value is converted to floating point before being passed to the plug-in.

Disabling the Apply Button

As a final step, your code should disable Apply in the property page dialog box as a signal to the user that the values have been successfully updated. This requires the following single line of code:
m_bDirty = FALSE; // Tell the property page to disable Apply.

See Also

- Modifying the Echo Sample Property Page

Implementing CEcho::FinalConstruct

The CEcho::FinalConstruct method is implemented in Echo.cpp. It contains code to read the property values from the registry when Windows Media Player instantiates the DSP plug-in object. This is important because it allows the user settings to persist between instances of the object, as well as between sessions. The plug-in wizard sample code provides implementation to read a single property from the registry. You can modify this code to handle the delay time property, and then add code to read the wet mix property value.

The following example code reads each property value from the registry and stores each in the correct member variable:

```cpp
CRegKey key;
LONG lResult;
DWORD dwValue;

lResult = key.Open(HKEY_CURRENT_USER, kszPrefsRegKey, KEY_READ);
if (ERROR_SUCCESS == lResult)
{
    // Read the delay time from the registry.
    lResult = key.QueryValue(dwValue, kszPrefsDelayTime);
    if (ERROR_SUCCESS == lResult)
    {
        m_dwDelayTime = dwValue;
    }

    // Read the wet mix value from the registry.
    lResult = key.QueryValue(dwValue, kszPrefsWetMix);
    if (ERROR_SUCCESS == lResult)
    {
        // Convert the DWORD to a double.
        m_fWetMix = (double)dwValue / 100;
        // Calculate the dry mix value.
        m_fDryMix = 1.0 - m_fWetMix;
    }
}

return S_OK;
```
Notice that the DWORD value for the wet mix is converted to a floating point value. Also note that the code calculates the correct value for m_fDryMix.

See Also

- Modifying the Echo Sample Property Page

Working with Streaming Resources

The sample audio DSP plug-in project generated by the Windows Media Player Plug-in Wizard does not require any streaming resources to be allocated by the plug-in. The Echo sample, however, requires a separate buffer to hold the audio data for a period of time to create the delay effect. The buffer is managed by two methods: IMediaObject::AllocateStreamingResources, which creates the buffer, and IMediaObject::FreeStreamingResources, which releases the buffer. The IMediaObject methods are implemented in Echo.cpp.

The following sections provide further information about managing the buffers:

- Variables to Manage the Delay Buffer
- Implementing IMediaObject::AllocateStreamingResources
- Implementing IMediaObject::FreeStreamingResources

See Also

- The Echo Sample

Variables to Manage the Delay Buffer
You must add the declarations for the member variables you need to manage the delay buffer. Add the following declarations to Echo.h in the `private` section:

```cpp
dWORD  m_cbDelayBuffer;   // Count of bytes in delay buffer size.
BYTE*  m_pbDelayPointer;  // Movable pointer to delay buffer.
BYTE*  m_pbDelayBuffer;   // Pointer to the head of the delay buffer.
```

Add the following code to the CEcho constructor to initialize the delay buffer variables:

```cpp
m_pbDelayBuffer = NULL;
m_pbDelayPointer = NULL;
m_cbDelayBuffer = 0;
```

**See Also**

- Working with Streaming Resources

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Implementing IMediaObject::AllocateStreamingResources**

In the Echo sample, the `AllocateStreamingResources` method creates the delay buffer. It does this by doing the following:

1. Calculating a size for the buffer that corresponds to the specified delay time for the given media type.
2. Either allocating new memory or reallocating the buffer size if it already exists.
3. Calling a method that fills the buffer with values representing silence.

The value for silence is different depending upon the bit depth. For 8-bit audio, silence is represented by the hex value 80; for 16-bit audio, silence is represented by zero.

**Calculating the Delay Buffer Size**

In order to calculate the size required for the delay buffer, you must first retrieve a `WAVEFORMATEX` structure that contains information about the audio data. The following example retrieves a pointer to this structure from the input `DMO_MEDIA_TYPE` structure:

```cpp
// Get a pointer to the WAVEFORMATEX structure.
WAVEFORMATEX *pWave = ( WAVEFORMATEX * ) m_mtInput.pbFormat;
if (NULL == pWave)
{
    return E_FAIL;
}
```
Once you have stored a pointer to the proper `WAVEFORMATEX` structure, you can inspect its members and use them to calculate the required buffer size. The following code example demonstrates this:

```cpp
// Get the size of the buffer required.
m_cbDelayBuffer = (m_dwDelayTime * pWave->nSamplesPerSec * pWave->nBlockAlign) / 1000;
```

This algorithm computes the buffer size by multiplying the delay time, in milliseconds, by the number of samples per millisecond, then multiplying the result by the number of channels, and then multiplying the result by the number of bytes per sample. The number of channels and the number of bytes per sample are not obvious; they are encoded in the `nBlockAlign` member. Dividing by 1000 reduces the `nSamplesPerSec` value to samples per millisecond; the millisecond is the desired unit because the delay time is expressed in milliseconds.

### Allocating the Delay Buffer Memory

Once you have calculated the memory requirements, you can allocate the buffer. The buffer may need to be deleted if it exists, such as when the user invokes the property page to change the delay time value. The following code demonstrates allocating the delay buffer:

```cpp
// Test whether a buffer exists.
if (m_pbDelayBuffer)
{
    // A buffer already exists.
    // Delete the delay buffer.
    delete m_pbDelayBuffer;
    m_pbDelayBuffer = NULL;
}

// Allocate the buffer.
m_pbDelayBuffer = new BYTE[m_cbDelayBuffer];
if (!m_pbDelayBuffer)
    return E_OUTOFMEMORY;
```

If the buffer is successfully allocated, you should move the movable pointer to the head of the buffer, as demonstrated in the following example:

```cpp
// Move the echo pointer to the head of the delay buffer.
m_pbDelayPointer = m_pbDelayBuffer;
```

### Filling the Delay Buffer with Silence

It is convenient to write a method to fill the delay buffer with values representing silence. The method should receive the pointer to the valid `WAVEFORMATEX` structure and then inspect the `wBitsPerSample` member to determine whether the audio is 8-bit or higher. The following example fills the delay buffer with the correct value for silence:

```cpp
void CEcho::FillBufferWithSilence(WAVEFORMATEX *pWfex)
{
    if (8 == pWfex->wBitsPerSample)
    {
        ::FillMemory(m_pbDelayBuffer, m_cbDelayBuffer, 0x80);
    }
    else
        ::ZeroMemory(m_pbDelayBuffer, m_cbDelayBuffer);
}
```
Remember to add the forward declaration for the function to the header file Echo.h in the private section:

```c
void FillBufferWithSilence(WAVEFORMATEX *pWfex);
```

You should add code at the end of `AllocateStreamingResources` to call this method each time the delay buffer is created or resized. The following example code passes the `WAVEFORMATEX` pointer named `pWave` to the new method:

```c
// Fill the buffer with values representing silence.
FillBufferWithSilence(pWave);
```

### Reallocating the Delay Buffer Memory

When the user changes the delay time by using the property page, the size of the delay buffer must change as well. You've already added the code to `AllocateStreamingResources` to resize the buffer, but you need to add a line of code to `CEcho::put_delay` to call the resource allocation method each time the property value changes. Here is the code:

```c
// Reallocate the delay buffer.
AllocateStreamingResources();
```

### See Also

- [Working with Streaming Resources](#)

---

### Implementing IMediaObject::FreeStreamingResources

It is important that your code releases any allocated memory before the plug-in object is destroyed. Windows Media Player calls `FreeStreamingResources` to allow you to do this. For safety, the sample created by the plug-in wizard includes a call to `FreeStreamingResources` in the `FinalRelease` method to ensure that the memory is freed. You must add the following code to `FreeStreamingResources` for the Echo sample:

```c
// Test whether a buffer exists.
if (m_pbDelayBuffer)
{
    delete m_pbDelayBuffer;
    m_pbDelayBuffer = NULL;
    m_pbDelayPointer = NULL;
    m_cbDelayBuffer = 0;
}
```

### See Also
Implementing CEcho::DoProcessOutput

The DoProcessOutput method performs the digital signal processing. This is the method that makes the changes to the data provided by Windows Media Player. It is the results of this method that you will hear as an echo effect when your Echo sample plug-in is complete.

For this sample, the plug-in will only process 8-bit or 16-bit audio. You will need to make some changes to the plug-in wizard sample code to remove the sections that process higher bit depth audio. It is worthwhile to study these sections, however, because you may decide to add your own echo implementation for those formats.

The following sections detain the changes you need to make to the code:

- Removing the Code to Process Greater than 16 Bits
- Processing the Audio Data
- Variables to Perform Processing
- Creating the Echo Effect

See Also

- The Echo Sample

Removing the Code to Process Greater than 16 Bits

Because this sample only processes 8-bit or 16-bit audio, you need to modify the code in CEcho::ValidateMediaType to return DMO_E_TYPE_NOT_ACCEPTED for media types greater than 16 bits. To accomplish this, you must change the code in the switch block that tests formats of type
WAVE_FORMAT_EXTENSIBLE. Replace the wizard code with the following example code:

```c
case WAVE_FORMAT_EXTENSIBLE:
    {
        // Sample size is greater than 16-bit or is multichannel.
        WAVEFORMATEXTENSIBLE *pWaveXT = (WAVEFORMATEXTENSIBLE *) pWave;
        if (KSDATAFORMAT_SUBTYPE_PCM != pWaveXT->SubFormat)
            return DMO_E_TYPE_NOT_ACCEPTED;
    } break;
```

Next, delete or comment out the sections of code in `DoProcessOutput` that handle high bit resolution audio. These are the sections that begin with case 24 and case 32.

**See Also**

- [Implementing CEcho::DoProcessOutput](#)

---

### Processing the Audio Data

The default implementation of `DoProcessOutput` begins by retrieving a pointer to a valid `WAVEFORMATEX` structure, exactly like was done in `AllocateStreamingResources`. It then uses the information in that structure to calculate the number of samples in the input buffer waiting to be processed. The following code is from the default implementation:

```c
// Get a pointer to the valid WAVEFORMATEX structure
// for the current media type.
WAVEFORMATEX *pWave = ( WAVEFORMATEX * ) m_mtInput.pbFormat;

// Calculate the number of samples to process.
DWORD dwSamplesToProcess = (*cbBytesProcessed / pWave->nBlockAlign) * pWave->nChannels;
```

Then, the code inspects the `wBitsPerSample` member to determine the bit depth of the audio. This value is used in a switch statement to provide separate processing for 8-bit and 16-bit audio.

**Differences Between 8-bit and 16-bit Audio**

There are important differences between 8-bit and 16-bit audio. Therefore, the processing routines to create the echo effect are different. The two formats differ in the following ways:
Each format has a different sample size: 8-bit samples each occupy one byte of memory, while 16-bit samples each occupy two bytes.
Each format represents the audio amplitude differently. 8-bit audio is represented by an unsigned integer with a range from 0 to 255; a value of 128 represents silence. 16-bit audio is represented by a signed integer with a range from -32768 to 32767; a value of zero represents silence.

While the process of creating the echo effect is fundamentally identical for each format, the details must differ slightly.

See Also

- Implementing CEcho::DoProcessOutput

Variables to Perform Processing

The member variables for handling the delay buffer deal with BYTE quantities; they are BYTE pointers and an integer that stores a count of bytes. This is ideal for processing 8-bit audio, since an 8-bit sample fits nicely into one byte of memory. When processing 16-bit audio, though, it is more convenient to convert these to short pointers, so the processing can occur two bytes at a time.

The following example code allocates the new 16-bit pointers, and adds a pointer variable that stores the address of the end of the delay buffer. Insert it in the case 16 section just before the loop entry point:

```cpp
// Store local pointers to the delay buffer.
short    *pwDelayPointer = (short    *)m_pbDelayPointer;
short    *pwDelayBuffer = (short    *) m_pbDelayBuffer;
// Store the address of the last word of the delay buffer.
short    *pwEOFDelayBuffer = (short    *)(m_pbDelayBuffer + m_cbDelayBuffer - sizeof(short));

// Store the address of the end of the delay buffer.
BYTE * pbEOFDelayBuffer = (m_pbDelayBuffer + m_cbDelayBuffer - sizeof(BYTE));
```

The code for 8-bit processing also allocates a variable that stores the address of the end of the delay buffer. Storing this value makes it easy to test whether the movable delay buffer pointer has reached the end of the delay buffer. The following example code calculates the value:

```cpp
// Store the address of the end of the delay buffer.
BYTE * pbEOFDelayBuffer = (m_pbDelayBuffer + m_cbDelayBuffer - sizeof(BYTE));
```

See Also

- Implementing CEcho::DoProcessOutput
Creating the Echo Effect

You must first remove the code from the wizard sample that scales the audio. From the 8-bit section, remove the following code:

```c
// Apply scale factor to sample.
i = int( ((double) i) * m_dwDelayTime );
```

(Remember that m_fScaleFactor was replaced by m_dwDelayTime.)

From the 16-bit section, remove the following code:

```c
// Apply scale factor to sample.
i = int( ((double) i) * m_dwDelayTime );
```

The implementation of `DoProcessOutput` provided by the plug-in wizard sample code creates a while loop that iterates one time for each sample in the input buffer provided by Windows Media Player. This loop works the same way for both 8-bit and 16-bit audio, although a separate loop is required for each. In each case, the loop initiates with the following test:

```c
while (dwSamplesToProcess--)
```

Once inside the loop, the processing routines are very similar for 8-bit and 16-bit audio. The main difference is that the code in the 8-bit section changes the range of data values to -128 through 127, and then converts the range back again before writing the data to the output buffer. This is important to retain the symmetry of the audio waveform during processing.

Now you can begin to add and replace code in the processing loop.

Retrieve a Sample from the Input Buffer

During each iteration of the loop, a single sample is retrieved from the input buffer. For 8-bit audio, the sample is shifted into the new range, and then the pointer to the input buffer is advanced to the next sample. The following code is from the plug-in wizard:

```c
// Get the input sample and normalize to -128 .. 127
int i = (*pInputData++) - 128;
```

For 16-bit audio, the process is the same except for the normalization:

```c
// Get the input sample.
int i = *pInputData++;
Remember that the pointers in the 16-bit code have been converted to type **short**.

### Retrieve a Sample from the Delay Buffer

Next, retrieve a single sample from the delay buffer. For 8-bit code, the delay samples are stored in their native range of 0 to 255. The following code, which you must add, retrieves an 8-bit delay sample:

```c
// Get the delay sample and normalize to -128 .. 127
int delay = m_pbDelayPointer[0] - 128;
```

For 16-bit audio, the process is similar:

```c
// Get the delay sample.
int delay = *pwDelayPointer;
```

### Write the Input Sample to the Delay Buffer

Now, you must store the input sample in the delay buffer in the same location from which you retrieved the delay sample. The following is the code you must add for 8-bit audio:

```c
// Write the input sample into the delay buffer.
m_pbDelayPointer[0] = i + 128;
```

This is the code to add for the 16-bit section:

```c
// Write the input sample to the delay buffer.
*pwDelayPointer = i;
```

### Move the Delay Buffer Pointer

Now that the work in the delay buffer is finished for this iteration, you can advance the movable pointer to the delay buffer. If the pointer reaches the end of the circular buffer, you must change its value to point to the head of the buffer. To do this for 8-bit audio, use the following code:

```c
// Increment the delay pointer.
// If it has passed the end of the buffer,
// then move it to the head of the buffer.
if (++m_pbDelayPointer > pbEOFDelayBuffer)
    m_pbDelayPointer = m_pDelayBuffer;
```

Here is the code for the 16-bit section:

```c
// Increment the local delay pointer.
// If it is past the end of the buffer,
// then move it to the head of the buffer.
if (++pwDelayPointer > pwEOFDelayBuffer)
    pwDelayPointer = pwDelayBuffer;
```

Since the pointer in the 16-bit section is really a copy of the member variable, you must remember to update the value in the member variable with the new address. If you fail to do this, the delay buffer pointer will point to the head of the buffer repeatedly and your echo effect will not work as expected. Add the following code to the 16-bit section:

```c
// Move the global delay pointer.
m_pbDelayPointer = (BYTE *) pwDelayPointer;
```
Mix the Input Sample with the Delay Sample

This is where you use the wet mix and dry mix values to create the final output sample. You simply multiply each sample by the floating point value that represents the percentage of the final signal for the sample. Multiply the input sample by the value stored in m_fDryMix; multiply the delay sample by the value stored in m_fWetMix. Then, add the two values. The code you must add is identical for the 8-bit and 16-bit sections:

```c
// Mix the delay with the dry signal.
i = (int)((i * m_fDryMix) + (delay * m_fWetMix));
```

Write the Data to the Output Buffer

Finally, copy the mixed sample to the output buffer, and then advance the output buffer pointer. For 8-bit audio, the plug-in wizard uses the following code to return the sample to its original range:

```c
// Convert back to 0..255 and write to output buffer.
*pbOutputData++ = (BYTE)(i + 128);
```

For 16-bit audio, the wizard uses the following code as the final step in the processing loop:

```c
// Write to output buffer.
*pwOutputData++ = i;
```

See Also

- [Implementing CEcho::DoProcessOutput](#)

Using the Echo Sample DSP Plug-in

When you create an error-free build of your Windows Media Player echo DSP plug-in, you can use it in Windows Media Player to hear the effect. When you start Windows Media Player, the plug-in should be enabled by default. If you play audio or video with audio, you should hear a one-second echo in the sound that plays at the same volume as the original sound.

You can experiment with different delay times and effects levels by changing the values in the echo plug-in property page. Try the effect on different types of source material, such as music or speech. A delay time of 200 ms with an effects level of 30 will result in a tightly spaced echo that sounds like the effect used on early rock and roll recordings. Effects levels greater than 50 will result in the original signal being lower in volume than the echo, which is called a pre-echo effect.
DSP Plug-ins Programming Reference

The Microsoft Windows Media Player Software Development Kit (SDK) supports a range of interfaces and methods for digital signal processing (DSP) plug-ins. The following section documents these in detail.

### Section

**DSP Plug-in Interfaces**

Lists the interfaces and methods used for DSP plug-ins.

**DSP Plug-in Enumeration Types**

Lists the enumeration types used for DSP plug-ins.

See Also

- [Windows Media Player DSP Plug-ins](#)

DSP Plug-in Interfaces

The digital signal processing (DSP) plug-in interfaces are used to manage data transfer between Windows Media Player and the plug-in. All DSP plug-ins may optionally implement the `ISpecifyPropertyPage` interface to provide a property page implementation.

The following interfaces are required for creating DSP plug-ins.
## IWMPMediaPluginRegistrar Interface

The **IWMPMediaPluginRegistrar** interface manages plug-in registration.

In addition to the methods inherited from **IUnknown**, the **IWMPMediaPluginRegistrar** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WMPRegisterPlayerPlugin</strong></td>
<td>Adds information to the registry that identifies a Windows Media Player plug-in.</td>
</tr>
<tr>
<td><strong>WMPUnRegisterPlayerPlugin</strong></td>
<td>Removes information from the registry about a Windows Media Player plug-in.</td>
</tr>
</tbody>
</table>

### See Also

- [DSP Plug-in Interfaces](#)
IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin

The **IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin** function adds information to the registry that identifies a Windows Media Player plug-in.

**Syntax**

```c
HRESULT WMPRegisterPlayerPlugin(
    LPWSTR pwszFriendlyName,
    LPWSTR pwszDescription,
    LPWSTR pwszUninstallString,
    DWORD dwPriority,
    GUID guidPluginType,
    CLSID clsid,
    UINT cMediaTypes,
    LPVOID pMediaTypes
);
```

**Parameters**

*pwszFriendlyName*

Pointer to a wide character null-terminated string containing the friendly name of the plug-in. This is also the name that displays to the user.

*pwszDescription*

Pointer to a wide character null-terminated string containing the description of the plug-in. This information also displays to the user.

*pwszUninstallString*

Pointer to a wide character null-terminated string containing the uninstall string.

*dwPriority*

Integer value containing the priority position of the plug-in in the chain of currently enabled plug-ins.

*guidPluginType*

GUID specifying plug-in type. For DSP plug-ins, specify WMP_PLUGINTYPE_DSP.

*clsid*


The class ID of the plug-in.

cMediaTypes

Count of media types supported by the plug-in.

pMediaTypes

Pointer to an array of media types that enumerates the supported media types. Media types are stored as type/subtype pairs.

Return Values

The function returns an HRESULT.

Remarks

Implement this function in the exported DllRegisterServer function.

The uninstall string is a command line string that Windows Media Player passes as the argument to the Windows ShellExecute function when the user chooses to remove the plug-in by clicking Remove in the Player plug-in configuration dialog box. This gives you a way to execute your own uninstall program that initiates from Windows Media Player.

Priority values start at zero. Most DSP plug-ins should specify a value between 1 and 10. Lower values place the plug-in closer to the rendering engine.

DSP plug-ins registered with identical values for dwPriority are ordered based on their position in the registry. Plug-ins located higher in the registry hierarchy are assigned a higher priority than plug-ins located lower in the registry hierarchy.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

Library: Use wmp.dll.

See Also

- IWMPMediaPluginRegistrar Interface
- IWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin
IWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin

The **IWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin** function removes information from the registry about a Windows Media Player plug-in.

Syntax

```c
HRESULT WMPUnRegisterPlayerPlugin(
    GUID guidPluginType,
    CLSID clsid
);
```

**Parameters**

*guidPluginType*

GUID specifying plug-in type. For DSP plug-ins, specify WMP_PLUGINTYPE_DSP.

*clsid*

Specifies the class ID of the plug-in being removed.

**Return Values**

The function returns an **HRESULT**.

**Remarks**

Implement this function in the exported **DllUnRegisterServer** function.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmpservices.idl; include wmpservices.h.

**Library**: Use wmp.dll.

**See Also**

- **IWMPMediaPluginRegistrar Interface**
- **IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin**

© 2000-2003 Microsoft Corporation. All rights reserved.
IWMPPlugin Interface

The IWMPPlugin interface is implemented by the plug-in. It manages the connection to Windows Media Player.

Note  The interface identifier GUID for this interface changed between the beta release and the final release.

In addition to the methods inherited from IUnknown, the IWMPPlugin interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdviseWMPServices</td>
<td>Receives a pointer to a Windows Media Player interface that contains methods that provide stream state information.</td>
</tr>
<tr>
<td>GetCaps</td>
<td>Sets a value that specifies whether the plug-in requires the input format and output format to be identical.</td>
</tr>
<tr>
<td>GetID</td>
<td>Returns the Class ID of the plug-in.</td>
</tr>
<tr>
<td>Init</td>
<td>Receives a playback context identifier.</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Executes when Windows Media Player shuts down the plug-in.</td>
</tr>
<tr>
<td>UnAdviseWMPServices</td>
<td>Executes when Windows Media Player releases the pointer provided in AdviseWMPServices.</td>
</tr>
</tbody>
</table>

See Also

- DSP Plug-in Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.
The **IWMPPlugin::AdviseWMPServices** method is implemented by the plug-in.

**Syntax**

```c
HRESULT AdviseWMPServices(
    IWMPServices*  pWMPServices
);
```

**Parameters**

- **pWMPServices**
  
  [in] Pointer to an **IWMPServices** interface.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

Windows Media Player calls the **AdviseWMPServices** method on the plug-in to pass in a pointer that the plug-in can then use to call the **IWMPServices** interface, which contains methods that provide information about the current state of the stream.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmpservices.idl; include wmpservices.h.

**See Also**

- **IWMPPlugin Interface**
- **IWMPServices Interface**
input format and an output format.

Syntax

```c
HRESULT GetCaps(
    DWORD* pdwFlags
);
```

Parameters

`pdwFlags`

[out] Pointer to a variable that specifies whether the plug-in can convert formats. The specified value is a bitwise combination of zero or more flags from the `WMPPlugin_Caps` enumeration.

Return Values

The method returns an `HRESULT`.

Remarks

There are currently two possible [out] values that the plug-in may specify: zero to indicate that the plug-in can convert formats, or `WMPPlugin_Caps_CannotConvertFormats`, which forces Windows Media Player to handle any necessary format conversion.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

See Also

- IWMPPlugin Interface
- WMPPlugin_Caps

IWMPPlugin::GetID

The `IWMPPlugin::GetID` method returns the class id of the plug-in.

Syntax
HRESULT GetID(  
    GUID* pGUID  
);  

Parameters

pGUID

[out] Pointer to a GUID that represents the class id of the plug-in.

Return Values

The method returns an HRESULT.

Remarks

For more information on the GUID structure, see the Platform SDK.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

See Also

● IWMPP{ }Interface

IWMPPlugin::Init

The IWMPPlugin::Init method is called when Windows Media Player initializes the plug-in.

Syntax

HRESULT Init(  
    DWORD dwPlaybackContext  
);  

Parameters

dwPlaybackContext
DWORD value that indicates the particular Windows Media Player playback engine to which the plug-in belongs.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

It is possible at any given time that multiple instances of Windows Media Player could be running in the same process. For instance, multiple Windows Media Player control instances could be embedded in the same browser window, or even in multiple instances of a browser that coexist in the same process. It is also possible that the same instance of Windows Media Player could create multiple playback engines at the same time. The `dwPlaybackContext` value allows you to determine which instance of the Windows Media Player playback engine contains the plug-in. This is useful if you wish to enable multiple plug-ins to connect to each other.

**Init** and **Shutdown** will always be called on the same thread.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpservices.idl; include wmpservices.h.

**See Also**

- **IWMPPlugin Interface**

---

The **IWMPPPlugin::Shutdown** method is called when Windows Media Player shuts down the plug-in.

**Syntax**

```cpp
HRESULT Shutdown();
```

**Return Values**

The method returns an **HRESULT**.
Remarks

Init and Shutdown will always be called on the same thread.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

See Also

- IWMPPPlugin Interface

IWMPPlugin::UnAdviseWMPServices

The IWMPPlugin::UnAdviseWMPServices method is used to release the pointer provided by AdviseWMPServices.

Syntax

HRESULT UnAdviseWMPServices();

Parameters

The method takes no parameters.

Return Values

The method returns an HRESULT.

Remarks

Windows Media Player calls this method when the pointer provided by AdviseWMPServices is no longer valid. The plug-in should use this method to cease making stream state requests through the pointer.

Requirements

Version: Windows Media Player 9 Series or later.
IWMPPluginEnable Interface

The **IWMPPluginEnable** interface is implemented by the plug-in. It sets and retrieves a value that represents whether the plug-in has been enabled by Windows Media Player.

In addition to the methods inherited from **IUnknown**, the **IWMPPluginEnable** interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetEnable</td>
<td>Retrieves the current enable state.</td>
</tr>
<tr>
<td>SetEnable</td>
<td>Sets the current enable state.</td>
</tr>
</tbody>
</table>

See Also

- DSP Plug-in Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.
The **IWMPPPluginEnable::GetEnable** method returns a value indicating whether Windows Media Player has enabled the plug-in.

**Syntax**

```c
HRESULT GetEnable(
    BOOL*   pfEnable
);
```

**Parameters**

*pfEnable*

[out] Pointer to a **Boolean** value indicating whether the user has enabled the plug-in.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

This value is set by Windows Media Player in the **IWMPPPluginEnable::SetEnable** method.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpservices.idl; include wmpservices.h.

**See Also**

- **IWMPPPluginEnable Interface**
- **IWMPPPluginEnable::SetEnable**
HRESULT SetEnable(
    BOOL  fEnable
);

Parameters

fEnable

[in] A variable that receives a value indicating whether the user has enabled the plug-in.

Return Values

The method returns an HRESULT.

Remarks

The value of fEnable depends upon whether the user has enabled the plug-in on the Plug-ins tab of the Options dialog box in Windows Media Player.

DSP plug-ins are removed from the signal path when the user chooses to disable them. When disabled, a plug-in should still include code to copy data from the input buffer to the output buffer, performing only format conversion processing, if applicable. The sample code generated by the Windows Media Player plug-in wizard includes code to copy the buffers.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

See Also

- IWMPPPluginEnable Interface
- IWMPPPluginEnable::GetEnable

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPservices Interface

The IWMPservices interface is implemented by Windows Media Player. It provides methods to retrieve the current stream state and current stream time.
In addition to the methods inherited from **IUnknown**, the **IWMPServices** interface exposes the following methods.

**Method**  
**GetStreamState**  
**GetStreamTime**

**Description**  
Returns a value that represents the current stream state.

Returns a value that indicates the current stream time.

**See Also**

- DSP Plug-in Interfaces

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPServices::GetStreamState**

The **IWMPServices::GetStreamState** method retrieves information about the current play state of the stream.

**Syntax**

```cpp
HRESULT GetStreamState(
    WMPServices_StreamState* pState
);
```

**Parameters**

**pState**

[in] A pointer to a **WMPServices_StreamState** enumeration value.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

The stream is stopped, paused, or playing.

**Requirements**

**Version:** Windows Media Player 9 Series or later.
IWMPServices::GetStreamTime

The **IWMPServices::GetStreamTime** method retrieves a structure indicating the current stream time.

**Syntax**

```c
HRESULT GetStreamTime(
    REFERENCE_TIME*  prt
);
```

**Parameters**

`prt`

[in] Pointer to a `REFERENCE_TIME` structure.

**Return Values**

The method returns an `HRESULT`.

**Remarks**

The current stream time is determined by Windows Media Player. This means that the value returned by this method do not necessarily represent the elapsed time relative to the beginning of the file. For instance, if the user moves the trackbar in the Player to seek the media to a new position, the value returned by this method returns the time elapsed since the media began playing from the new position. Changes in playback rate will also affect the value returned by this method.

The values provided in the `rtTimestamp` member of `IMediaObject::ProcessInput` and the `rtTimestamp` member of the `DMO_OUTPUT_DATA_BUFFER` structure supplied by `IMediaObject::ProcessOutput` contain values that indicate when the data provided in the buffer will be rendered relative to the current stream time. Therefore, these values also do not necessarily represent the elapsed time relative to the beginning of the
file file or the presentation time specified in the file.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

Library: Use wmp.dll.

See Also

- **IWMPServices Interface**

DSP Plug-in Enumeration Types

The Windows Media Player SDK implements the following enumeration type for creating DSP plug-ins.

<table>
<thead>
<tr>
<th>Enumeration Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMPPlugin_Caps</td>
<td>Used with IWMPPushin::GetCaps to indicate whether the plug-in can convert between formats.</td>
</tr>
<tr>
<td>WMPServices_StreamState</td>
<td>Indicates the whether the stream is currently stopped, paused, or playing.</td>
</tr>
</tbody>
</table>

See Also

- **DSP Plug-ins Programming Reference**
The **WMPPlugin_Caps** enumeration type signals whether the plug-in can convert between input and output formats.

**Syntax**

```c
enum WMPPlugin_Caps{
    WMPPlugin_Caps_CannotConvertFormats = 1
};
```

**Members**

**WMPPlugin_Caps_CannotConvertFormats**

The plug-in requires that the input format and output format be the same.

**Remarks**

When `IWMPPlugin::GetCaps` returns **WMPPlugin_Caps_CannotConvertFormats**, Windows Media Player handles any necessary format conversion.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpservices.idl; include wmpservices.h.

**See Also**

- DSP Plug-in Enumeration Types
- `IWMPPlugin::GetCaps`

© 2000-2003 Microsoft Corporation. All rights reserved.
WMPServices_StreamState

The WMPServices_StreamState enumeration indicates whether the stream is currently stopped, paused, or playing.

Syntax

```c
enum WMPServices_StreamState{
    WMPServices_StreamState_Stop = 0,
    WMPServices_StreamState_Pause = 1,
    WMPServices_StreamState_Play = 2
} WMPServices_StreamState;
```

Members

WMPServices_StreamState_Stop

The stream is stopped.

WMPServices_StreamState_Pause

The stream is paused.

WMPServices_StreamState_Play

The stream is playing.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

See Also

- DSP Plug-in Enumeration Types
- IWMPServices::GetStreamState
Microsoft Windows Media Player provides an architecture that enables you to develop plug-ins that decode (if necessary) and render custom data contained in a Windows Media format stream. Rendering plug-ins are DirectX Media Objects (DMOs) that connect to the Player using COM interfaces. A typical rendering plug-in might display animation video encoded in a custom format. This section of the SDK provides you with the programming information you need to create your own rendering plug-in.

The rendering plug-in documentation is divided into three sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Rendering Plug-ins</td>
<td>Provides an overview of the architecture used for rendering plug-ins. Read this section to learn the general concepts involved with this technology.</td>
</tr>
<tr>
<td>Rendering Plug-ins Programming Guide</td>
<td>Explains what you need to do to create a rendering plug-in. This section contains example code and step-by-step procedures.</td>
</tr>
<tr>
<td>Rendering Plug-ins Programming Reference</td>
<td>Provides a detailed reference for the COM interfaces, methods, and enumerated types supported by the Player SDK for rendering plug-ins.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player Plug-ins

© 2000-2003 Microsoft Corporation. All rights reserved.

About Rendering Plug-ins

Microsoft Windows Media Player provides an architecture for connecting add-on software that renders arbitrary data streams, which can be contained in Windows Media files, during playback. These add-on programs are referred to as rendering plug-ins.

The following sections provide a conceptual overview of Windows Media Player rendering plug-ins.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendering Plug-in User Overview</td>
<td>Describes rendering plug-ins from the end user's perspective.</td>
</tr>
<tr>
<td>Rendering Plug-in Developer Overview</td>
<td>Provides general information you need to know</td>
</tr>
</tbody>
</table>
To the user, rendering plug-ins are add-on features that enable Windows Media Player to render content in a Windows Media file that ordinarily wouldn't be available using Windows Media Player alone. The result of a rendering plug-in could be something obvious, like a custom animation video being displayed, or something subtle, like new functionality that lets the user click on a video image to change the playback experience.

Users can install and uninstall rendering plug-ins by using the Plug-ins tab of the Options dialog box in Windows Media Player. Rendering plug-ins may implement a property page, which users can also access from the Plug-ins tab. A property page provides the user with a way to change settings that affect the way a rendering plug-in works. For example, a property page might include radio buttons that allow the user to choose a background color.

Windows Media Player can display a small graphic, or series of graphics, in the status area of the Player to indicate to the user that a rendering plug-in is active. This graphic is provided to the Player by the plug-in if the plug-in supports this feature.
Rendering Plug-in Developer Overview

From the developer's perspective, rendering plug-ins are software programs that receive arbitrary data from Windows Media Player, then render that data.

The following sections provide conceptual information you should understand before developing a rendering plug-in for Windows Media Player:

- About Arbitrary Data Streams
- Rendering Plug-in Packaging
- Connecting to Windows Media Player
- Data Input and Rendering
- Format Negotiation
- Providing a User Interface

See Also

- About Rendering Plug-ins

About Arbitrary Data Streams

Arbitrary data is any data in any format. In the context of the Windows Media Player SDK, this means data in a format that is not supported by Windows Media Player. In the context of the Player, arbitrary data is simply binary data that the Player is not designed to play back natively.

The Windows Media Format SDK supports inserting arbitrary data into an arbitrary data stream by using Component Object Model (COM) objects provided by the SDK. Such a stream is identified by a GUID that uniquely identifies the media type of the data in the stream. This means that you can use arbitrary data streams to contain data in virtually any format, including formats you invent yourself, and then use a Windows Media Player rendering plug-in to display that data in the Player.

Note: Streaming Windows Media files that contain arbitrary data streams is supported in Windows Media Services 9 Series or later.
Note Arbitrary data samples to be rendered by Windows Media Player rendering plug-ins must not exceed 10 MB in size.

Windows Media Player will not instantiate a rendering plug-in to handle an arbitrary data stream, or play the content, unless the Windows Media file has a .asf or .wm file name extension. For more information about creating Windows Media files, download the Windows Media Format SDK from the Microsoft Web site.

See Also

- Rendering Plug-in Developer Overview

Warning Rendering plug-in objects must not be created as singletons. Windows Media Player must be able to create multiple separate instances of a particular rendering plug-in object.
Connecting to Windows Media Player

When attempting to play content containing an arbitrary data stream, Windows Media Player automatically connects to rendering plug-ins that have been installed and properly registered. Rendering plug-ins must call `IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin` to create the registry entries necessary to allow Windows Media Player to recognize the plug-in and to list it on the Plug-ins tab of the Options dialog box. To remove the registry entries created by `IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin`, the plug-in calls `IWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin`.

See Also

- [Rendering Plug-in Developer Overview](#)

Data Input and Rendering

Windows Media Player provides arbitrary data to rendering plug-ins through an input buffer allocated by the Player. Rendering plug-ins can then render data in the Windows Media Player video display region by using resources obtained during a prior negotiation with Windows Media Player. If Windows Media Player cannot provide space in its user interface for rendering, as in the case of an arbitrary data video stream combined with Windows Media-based video, then the plug-in must render in a separate window. Windows Media Player manages this negotiation by calling methods implemented by the plug-in.

In order to render data, a rendering plug-in requires a rectangle that defines the region on the Windows desktop in which the plug-in can draw, and a handle to a drawing resource. The plug-in must implement `IWMPNodeRealEstate::GetDesiredSize` in order to provide the Player with a pointer to a SIZE structure that represents the requested size of the rendering region required by the plug-in. In Windows, drawing rectangles are represented by RECT structures. Windows Media Player provides the plug-in three such structures by calling `IWMPNodeRealEstate::SetRects`. Together, these RECTs are used to calculate the dimensions and location of the rendering region provided by the Player. It is not guaranteed that requesting a particular size will result in the Player providing a rendering RECT of that size. The actual dimensions and location of the rendering region provided by the Player are determined by a variety of factors, such as whether the user has resized the user interface, whether the rendering is occurring in a skin or a Web page, and how the user has specified settings for sizing behaviors in the Player.

Because Windows Media Player can render in windowed or windowless mode, the plug-in must be able to
render data using either a window handle or a device context handle. In either case, the plug-in should create its
own window when instantiated. When rendering in windowed mode, this is the window where rendering takes
place. When rendering in windowless mode, this window is hidden.

Windows Media Player calls **IWMPNodeRealEstate::SetWindowless** to notify the plug-in about which
drawing state is the current one. When rendering in a window, the Player calls the plug-in implementation of
**IWMPNodeWindowed::SetOwnerWindow** to provide a window handle for rendering. The plug-in sets this
handle as the parent for the plug-in window.

Windows Media Player supplies data from the arbitrary data stream to the plug-in by calling
**IMediaObject::ProcessInput**, passing a pointer to an **IMediaBuffer** object. The plug-in must keep a reference
count on the buffer object until finished rendering the data in the buffer. Whenever the plug-in accesses the data
in the buffer object, it must do so within a critical section to ensure that the Player does not change the data on a
different thread.

Once the plug-in has a reference to the data buffer, it causes the drawing region to be invalidated in order to
prompt a repainting of the region. In windowed mode, the plug-in calls the Windows GDI method
**InvalidateRect**, followed by a call to **UpdateWindow**, which sends a WM_PAINT message to the rendering
window. The plug-in handles the WM_PAINT message in its rendering window and then renders the data. In
windowless mode, the plug-in must call **IWMPNodeWindowlessHost::InvalidateRect**. This causes Windows
Media Player to invalidate the rendering region and then call the plug-in implementation of
**IWMPNodeWindowless::OnDraw**. This method receives a device context handle and a pointer to a RECT
structure for rendering. The plug-in can then use these to render the data. Since rendering the data requires
access to the data buffer, rendering must take place within a critical section.

Rendering plug-ins don't provide an implementation for **IMediaObject::ProcessOutput** like other DMOs do.
Simply returning E_NOTIMPL is sufficient because the Player doesn't require and can't accept any processed
data from the plug-in. It is the job of the plug-in to perform all rendering functions.

See Also

- Rendering Plug-in Developer Overview

Format Negotiation

Since the format being used by a rendering plug-in is proprietary, format negotiation is very simple. Windows
Media Player doesn't need to work with the arbitrary data except to deliver it unchanged to the plug-in. When
the Player opens a Windows Media file that contains an arbitrary data stream, it retrieves the GUID of the
stream and checks the registry to locate a rendering plug-in registered with the matching GUID. When it finds
one, it loads the plug-in and calls **IMediaObject::SetInputType**, passing as an argument a
DMO_MEDIA_TYPE structure. The plug-in can use the members of this structure to validate that it supports
rendering the arbitrary data. At a minimum, the plug-in should verify that the major type member matches the GUID of the media type the plug-in is designed to process.

See Also

- Rendering Plug-in Developer Overview

Providing a User Interface

Rendering plug-ins can provide a property page in order to create a user interface. To do this, the plug-in must include a property page object that provides an implementation of an IPropertyPage interface. The rendering plug-in object must implement ISpecifyPropertyPages::GetPages, which allows Windows Media Player to locate and identify the correct property page for the plug-in.

Displaying a Status Graphic

Rendering plug-ins can display a small graphic, or series of graphics, in the Windows Media Player status area to notify the user that a plug-in is active. To support this feature, the plug-in must implement the IPropertyBag interface. Windows Media Player calls IPropertyBag::Read, providing a pointer to the requested property name "IconStreams", which is case-sensitive, and a pointer to a VARIANT structure that receives the data for the graphic. The plug-in creates an IStream object (or a SAFEARRAY of IStream objects if there are multiple graphics), then loads the graphic data, including header information, into the stream, and then returns a pointer to the IStream object using the punkVal member of the VARIANT structure. If the plug-in only supplies one graphic, it specifies the vt member of the VARIANT structure as VT_UNKNOWN. If the plug-in supplies multiple graphic IStream objects using a SAFEARRAY, it specifies the vt member of the VARIANT structure as VT_ARRAY.

Graphics can be stored in a variety of file formats, including:

**BMP**

Microsoft Windows Bitmap images are uncompressed.

**JPEG**

Compressed image format commonly used for Web pages. JPEG format files usually have .jpg file name extensions.

**GIF**
Compressed image format commonly used for Web pages.

**PNG**

Compressed image format commonly used for Web pages.

The maximum dimensions for rendering plug-in graphics are 38 pixels wide and 14 pixels high.

The **IStream** byte stream containing the status graphic must include header information. Without header information, Windows Media Player cannot properly identify the type of graphic and therefore will not load the image.

See Also

- Rendering Plug-in Developer Overview

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

### Rendering Plug-in Wizard

The Windows Media Player SDK provides a COM wizard that you can add to Visual C++ 6.0 or Visual C++ .NET that will generate the code for a rendering plug-in. This code includes sample method implementations that render text data from a sample digital media file. You can compile and link this sample code, and then register the resulting DLL file to try the sample rendering plug-in. You can then modify the generated code to create your own rendering plug-in. For more information, see Building a Rendering Plug-in.

See Also

- Rendering Plug-in Developer Overview

---

© 2000-2003 Microsoft Corporation. All rights reserved.
## Required Interfaces

A Windows Media Player rendering plug-in must implement certain interfaces. The following sections provide an overview of these interfaces.

### Interface

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About IMediaObject</strong></td>
<td>Required interface for a DirectX Media Object (DMO). Detailed documentation for this interface is located in the DirectX 8 SDK.</td>
</tr>
<tr>
<td><strong>About IWMPNodeRealEstate</strong></td>
<td>Makes requests and receives responses related to the rendering region.</td>
</tr>
<tr>
<td><strong>About IWMPNodeWindowed</strong></td>
<td>Specifies and retrieves the owner window.</td>
</tr>
<tr>
<td><strong>About IWMPNodeWindowless</strong></td>
<td>Initiates drawing in windowless mode.</td>
</tr>
<tr>
<td><strong>About IWMPPlugin</strong></td>
<td>Manages the connection to Windows Media Player.</td>
</tr>
<tr>
<td><strong>About IWMPWindowMessageSink</strong></td>
<td>Receives Windows messages when the plug-in is in windowless mode.</td>
</tr>
</tbody>
</table>

In addition to these interfaces, the plug-in can implement any additional interfaces required to do the job. For instance, DMOs can implement several interfaces specific to the DMO architecture. By implementing the DMO interfaces as required, you can create a plug-in that functions both as a DMO and as a Windows Media Player plug-in.

### See Also

- Rendering Plug-in Developer Overview
- Rendering Plug-ins Interfaces

### About IMediaObject

The **IMediaObject** interface is the required interface for DMOs. **IMediaObject** contains the methods that a Windows Media Player rendering plug-in uses to get data from the Player. For complete documentation of the **IMediaObject** interface, see the DirectX SDK.

The methods of **IMediaObject** can be categorized as follows:
Methods that Handle Format Negotiation

These are the methods that Windows Media Player calls to get information about the data formats supported by the plug-in. These methods include:

- `GetInputMaxLatency`
- `GetInputSizeInfo`
- `GetInputStreamInfo`
- `GetInputType`
- `GetOutputSizeInfo`
- `GetOutputStreamInfo`
- `GetStreamCount`
- `SetInputMaxLatency`
- `SetInputType`
- `SetOutputType`

Of these methods, only `SetInputType` is called for rendering plug-ins.

Methods that Specify or Retrieve State Information

These are the methods that a DMO client calls to get or set values related to the current state of the plug-in. These methods include:

- `GetInputCurrentType`
- `GetInputStatus`
- `GetOutputCurrentType`

Windows Media Player doesn't call these methods for rendering plug-ins.

Methods that Handle Buffering and Processing Data

These are the methods that Windows Media Player calls to initiate the various processes that the plug-in performs to do the digital signal processing. These methods include:

- `AllocateStreamingResources`
- `Discontinuity`
- `Flush`
- `FreeStreamingResources`
- `Lock`
- `ProcessInput`
- `ProcessOutput`

Windows Media Player calls `AllocateStreamingResources` and `FreeStreamingResources` to provide the rendering plug-in with an opportunity to set up or release any additional buffers the plug-in may require for internal processing.

Windows Media Player may call `Discontinuity` at any time to indicate an interruption in the stream. Rendering plug-ins can simply return S_OK.

Windows Media Player calls `Flush` to direct the rendering plug-in to flush all internally buffered data. The plug-in should release any references to `IMediaBuffer` interfaces, clear any values that specify the time stamp or sample length for the media buffer, and reinitialize any internal states that depend upon the contents of the media sample.
Windows Media Player calls `ProcessInput` to pass a pointer to an `IMediaBuffer` interface to the rendering plug-in. This interface provides access to the input buffer allocated by Windows Media Player to supply data to the plug-in.

The `IMediaObject` interface includes a method named `Lock`. This method is designed to acquire or release a lock on the DMO to keep the DMO serialized when performing multiple operations. The version of `IMediaObject::Lock` in the wizard code overrides the ATL implementation of `Lock`. Because the sample code is apartment threaded, the implementation of `Lock` simply returns `S_OK`. For details about how to create a multi-threaded DMO, refer to the DirectX SDK.

See Also

- Required Interfaces

About IWMPNodeRealEstate

The `IWMPNodeRealEstate` interface handles the rendering region negotiation between Windows Media Player and the plug-in. `IWMPNodeRealEstate::GetDesiredSize`, when called, requests a particular size rendering region from Windows Media Player. The Player will not necessarily provide a region of the requested size, but will attempt to do so. The actual rendering region available is conveyed to the plug-in in `IWMPNodeRealEstate::SetRects` in the form of three `RECT` structures: a source `RECT`, a destination `RECT`, and a clipping `RECT`. Using these three `RECTs`, the plug-in can calculate its own `RECT` for rendering.

`IWMPNodeRealEstate::SetWindowless` notifies the plug-in about the current rendering mode of the Player: windowed or windowless. Each of these modes requires a different method of initiating a repainting of the rendering region. Similarly, `IWMPNodeRealEstate::SetFullScreen` notifies the plug-in when the Player transitions to full screen mode.

See Also

- Required Interfaces
About IWMPNodeWindowed

The **IWMPNodeWindowed** interface notifies the plug-in about the parent window. **IWMPNodeWindowed::SetOwnerWindow** provides the plug-in with the window handle of the parent window. The plug-in window must specify this window as its own parent window. Conversely, when a NULL handle is supplied, the plug-in window must specify the desktop as its parent window. The plug-in should always return this value in **IWMPNodeWindowed::GetOwnerWindow**.

See Also
- Required Interfaces

About IWMPNodeWindowless

The **IWMPNodeWindowless** interface includes one method. Windows Media Player calls **IWMPNodeWindowless::OnDraw** to prompt the plug-in to render data when in windowless mode. Your implementation can use whatever Windows drawing technologies you choose to render your data.

See Also
- Required Interfaces
About IWMPPlugin

Windows Media Player calls the **IWMPPlugin** methods to provide the plug-in with information about the state of the connection between the plug-in and the Player. The Player calls **IWMPPlugin::Init** and **IWMPPlugin::Shutdown** to notify the plug-in about when the Player is connecting to and disconnecting from the plug-in. Rendering plug-ins also use **IWMPPlugin::Init** and **IWMPPlugin::Shutdown** to create and destroy the rendering window because these calls will always happen on the same thread. **Init** also provides a playback context id to allow the plug-in to determine which DirectShow graph contains it. **IWMPPlugin::AdviseWMPServices** and **IWMPPlugin::UnAdviseWMPServices** manage a pointer to **IWMPServices** to allow the plug-in to call methods on that interface. The Player can retrieve the class id of the plug-in by calling **IWMPPlugin::GetID**.

See Also

- [Required Interfaces](#)

About IWMPWindowMessageSink

When the plug-in is active without a window hosted in the Player user interface, such as when the Player is rendering Windows Media video or when in windowless mode, it requires a mechanism to receive window messages like mouse and keyboard events. Windows Media Player sends these messages to the plug-in by calling **IWMPWindowMessageSink::OnWindowMessage**.

Windows Media Player queries all active rendering plug-ins to determine whether they support this interface and automatically forwards window messages to each plug-in that does.

See Also

- [Required Interfaces](#)
Registering Rendering Plug-ins

Like other COM DLLs, you must register Windows Media Player rendering plug-ins to make them useable. Typically, use the regsvr32.exe utility to manually perform registration tasks. The Windows Media Player Plug-in Wizard creates files that contain registration scripts. These files have an .rgs file name extension. You can also create a setup program that copies your files and registers the DLLs.

You must invoke the registration methods on the IWMPMediaPluginRegistrar interface in the implementation blocks of the DllRegisterServer and the DllUnregisterServer functions, which are functions that are exported by COM DLLs. The Windows Media Player registration methods perform the necessary registration housekeeping to allow Windows Media Player to recognize your DLL file as a Windows Media Player plug-in so that it can be presented as an option to the user.

See Also

- IWMPMediaPluginRegistrar Interface
- Rendering Plug-in Developer Overview

Building a Rendering Plug-in

You can create a Windows Media Player rendering plug-in by using the Windows Media Player Plug-in Wizard. The Plug-in Wizard is included with the Windows Media Player SDK, which also includes this documentation. If you haven't installed the entire Windows Media Player SDK, you should do so before proceeding.

The Windows Media Player Plug-in Wizard is an add-in for Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET that creates Visual C++ projects and generates sample code for each of the different types of plug-ins supported by Windows Media Player.

The following sections explain what you need to know to create a rendering plug-in using the wizard:

- Getting Started with Rendering Plug-ins
- Using the Rendering Plug-in Wizard with Visual C++ 6.0
- Using the Rendering Plug-in Wizard with Visual C++ .NET
- About the Sample Rendering Plug-in
Getting Started with Rendering Plug-ins

To set up your development environment for creating rendering plug-ins, you must install the following items:

- Microsoft Visual C++ 6.0 or later
- Windows Media Player 9 Series
- Windows Media Player 9 Series SDK
- Microsoft DirectX 8 SDK
- Windows Media Player Plug-in Wizard

Installing Visual C++

You can install Visual C++ 6.0 or later by itself or as part of Microsoft Visual Studio. For help with installation, refer to the instructions that come with the Visual C++ or Visual Studio development environment.

Installing Windows Media Player

Install Windows Media Player 9 Series or any later version. You will need to use it to test your rendering plug-in.

Installing the Windows Media Player SDK

Be sure to install Windows Media Player 9 Series SDK or any later version of the SDK. In addition to this documentation, it includes the Windows Media Player Plug-in Wizard, as well as useful samples.

Installing the Microsoft DirectX 8 SDK

Install version 8.0 or newer of the Microsoft DirectX SDK. You'll need this to get the proper DMO headers and to link with msdmo.lib, which is a required library file for the code that the wizard generates.

**Note**  The Microsoft Windows XP SDK includes the DirectX 8 SDK files. If you have already installed the Windows XP SDK, you don't have to install the DirectX 8 SDK.

Installing the Plug-in Wizard
There are separate versions of the Windows Media Player Plug-in Wizard for Microsoft Visual C++ 6.0 and Microsoft Visual C++ .NET.

Once you have installed the Windows Media Player SDK, you can find the Windows Media Player Plug-in Wizard for Visual C++ 6.0 in the following subfolder:

\wizards\wmpplugin

The wizard file is named wmpwiz.awx.

Copy the wizard file to the following subfolder:

\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\IDE

The preceding path assumes you installed Microsoft Visual Studio 6.0 to the default location. If you installed the development environment to a different location, adjust the path accordingly. To quickly locate the correct subfolder, you can use Windows Explorer to search for files with an .awx file name extension.

If you are using Microsoft Visual C++ .NET, the Windows Media Player Plug-in Wizard is installed and configured automatically for you when you install this SDK. If you install Visual C++ .NET after installing this SDK, you should remove the SDK and reinstall it.

See Also

- Building a Rendering Plug-in

Using the Rendering Plug-in Wizard with Visual C++ 6.0

After you have installed the necessary components, creating a sample rendering plug-in is easy. The following steps will guide you:

2. From the File menu, click New…
4. In the Project name box, type a name for your rendering plug-in.
5. In the Location box, provide the location of a folder where the wizard can save the files it generates. You can accept the default, type a new path, or browse to an existing location.
6. Click OK.
7. Click Rendering Plug-in.
8. Click Next.
9. Click OK.

The wizard generates a complete rendering plug-in project.

Before attempting to build the project, be sure to configure your development environment to point to the folders named Include and Lib where you installed the DirectX 8 SDK. The compiler and linker will need access to some of the files in these folders. In Visual C++ 6.0, you can add these paths on the Directories tab of the Options dialog box, which you can access from the Tools menu.

See Also

- Building a Rendering Plug-in

© 2000-2003 Microsoft Corporation. All rights reserved.

Using the Rendering Plug-in Wizard with Visual C++ .NET

Once you have installed the necessary components, creating a sample rendering plug-in is easy. The following steps will guide you:

1. Start Microsoft Visual C++ .NET.
2. From the File menu, point to New, and then click Project.
3. In Project Types, click Visual C++ Projects if it isn't already selected.
4. In Templates, click Windows Media Player Plug-in Wizard to select it.
5. Type a name for your project.
6. Specify a location for your project. This is the folder to which your project files will be copied.
7. Click OK to start the wizard.
8. Click Rendering Plug-in.
9. Click Next.

The wizard generates a complete rendering plug-in project.

Before attempting to build the project, be sure to configure your development environment to point to the folders named Include and Lib where you installed the DirectX 8 SDK. Your compiler and linker will need access to some of the files in these folders. In Visual C++ .NET, you can add these paths on the VC++ Directories page of the Projects folder. The Projects folder is accessible from the Options dialog box, which you can access from the Tools menu.
About the Sample Rendering Plug-in

The sample rendering plug-in is designed to work together with the sample digital audio file named rendering.asf, which plays the same music as laure.wma. You can find the sample digital media file in the following folder where you installed this SDK:

\samples\media

The sample file contains an arbitrary data stream that uses a GUID to identify the media type of the data in the stream.

Warning  This GUID was generated specifically for the sample file; do not use it in your projects.

The GUID for the data in the sample file is defined as:

0xc1ccdf59, 0x6924, 0x4b96, 0x82, 0x47, 0xdb, 0xb0, 0xea, 0xe5, 0xb6, 0x7

To use the rendering plug-in that the Plug-in Wizard generates with the sample digital media file, you must replace the GUID in the definition created by the wizard with the preceding definition and then rebuild the project. The GUID definition can be found in the header file having the same name as your project. Rebuilding the project automatically registers the plug-in DLL on your computer.

The arbitrary data contained in rendering.asf is simply text data. The stream contains three lines of text. The first line is specified to display during playback at one second, the second line at six seconds, and the third line at 11 seconds. The sample plug-in creates a bitmap in memory, paints the background white, and then draws the text to the bitmap before rendering.

To see the sample text rendering, open Windows Media Player 9 Series after you've registered the sample plug-in. Play the sample file named rendering.asf.

The plug-in provides a sample property page implementation that allows the user to select the color of the displayed text.

The plug-in provides a sample logo graphic implementation as well.

See Also

- Building a Rendering Plug-in
Implementing Your Rendering Code

Once you have built the sample rendering plug-in, you can modify the code to create your own Windows Media Player rendering plug-in. Which methods you change and which you can leave as they are depends upon several factors, including:

- The data your plug-in is designed to render. If your data represents something visual, like video, then you'll want to provide a visible window for rendering like the sample generated by the wizard. If your data represents something else, like MIDI notes or hot spot coordinates, then you might never need to make the plug-in window visible. This would require changing the sample code.
- The number of properties you want to allow the user to change. You will certainly want to change the default property page implementation to suit your needs, and you may need to add additional properties.
- Whether your rendering plug-in needs to allocate any streaming resources. Your plug-in may require additional buffers.
- Whether your rendering plug-in needs to render in the Windows Media Player user interface (UI) Now Playing tab or in an external window.
- The number of status graphics you want your plug-in to display. The wizard-generated sample provides a single status graphic. You'll need to provide your own art if you include a graphic. Your plug-in can provide a series of graphics. Your plug-in is not required to include a graphic at all.

This section uses the rendering plug-in sample code generated by the Windows Media Player Plug-in Wizard to illustrate important concepts. You might find it helpful to open Microsoft Visual Studio and generate the sample code first so you can refer to it as you read this section. For details about how to use the Windows Media Player Plug-in Wizard, see Building a Rendering Plug-in.

Which changes you make to the code generated by the plug-in wizard depends on your particular needs. The range of possible applications for rendering plug-in technology is virtually limitless since the data you encode in your Windows Media file can be in any format and your plug-in can process the data any way you choose. However, you should take the time to understand how the code generated by the wizard works. That way, you'll know exactly which method implementations you should modify to get the result you seek.

The following sections provide details about how to modify the code generated by the Windows Media Player Plug-in Wizard to create your own rendering plug-in:

- About the Media Type GUID Definition
- About SetRects
- Implementing ProcessInput
- When to Render
About the Media Type GUID Definition

The very first code you'll probably need to modify is the definition of the GUID that represents your media type. Each time you generate a new sample rendering plug-in project using the Windows Media Player Plug-in Wizard, the wizard creates a new GUID for the project media type. You are certainly free to use this GUID for creating your content, but if you have already done that you'll need to change the GUID in the plug-in.

The media type GUID definition can be found in the header file having the same name as the plug-in project. For example, if you named the project MyRenderingPlugin, then the header file you need to change is named MyRenderingPlugin.h. When you view the file contents in Visual Studio, you'll see the GUID definition near the top of the file after the last include statement. It looks like this:

DEFINE_GUID(MEDIATYPE_MYRENDERINGPLUGIN, 0xc1ccdf59, 0x6924, 0x4b96, 0x82, 0x47, 0xdb, 0xb

Note   The GUID shown here is the media type identifier for the sample rendering file provided with this SDK. The GUID in your project will be different and unique.

If you already have a GUID for your media type, you must replace the provided GUID definition with your own. Be sure to use the DEFINE_GUID format like the sample code does.

See Also

- Implementing Your Rendering Code
About SetRects

When a plug-in renders in windowed mode and the rendering region is hosted in the Windows Media Player user interface, the Player must convey to the plug-in a set of values that describe the size and location of the portion of the rendering region that must be updated by the plug-in. The plug-in requests a particular region size when the Player calls the plug-in implementation of `IWMPNodeRealEstate::GetDesiredSize` by returning a pointer to a `SIZE` structure representing the desired size for the rendering region. The Player attempts to honor this request to the extent possible, but there is no guarantee that the ultimate region size will equal the requested size, so the plug-in must account for this possibility.

The Player notifies the plug-in about the actual region to be updated by calling the plug-in implementation of `IWMPNodeRealEstate::SetRects`. Here, the Player provides three pointers to `RECT` structures: `pSrc`, `pDest`, and `pClip`. For clarity, this section refers to these three `RECT`s as source rectangle, destination rectangle, and clipping rectangle. Together, these three rectangles are used to resize and to reposition the plug-in window so that it exactly corresponds to the region to be updated. That way, the plug-in can update the rendering region by painting in its window.

The source rectangle represents the portion of the original image to be rendered. The size of this rectangle does not have to equal the dimensions of the original area requested by the plug-in. The origin of the source rectangle can be any set of values specified by the Player. This rectangle is stored in the member variable named `m_rctSrc`. The following diagram shows the relationship between the video frame and the source rectangle.

![Source rectangle and video frame](image)

The destination rectangle is provided with an origin that represents positioning relative to the owner window and a size that represents stretching or shrinking relative to the source rectangle. This means that the destination rectangle is proportional to the source rectangle, but may be smaller, larger, or the same size, depending on conditions in the Player UI. The destination rectangle origin will always consist of a pair of values each greater than or equal to zero. This rectangle is stored in the member variable named `m_rctDest`.

The clipping rectangle represents the portion of the Windows Media Player video display that should actually be visible to the user. This rectangle is stored in the member variable named `m_rctClip`.

You can use these three rectangles to determine the size of the plug-in window and where to position it. To accomplish this, the rendering plug-in generated by the wizard creates two additional rectangles (each in a
RECT structure) using the information conveyed in the three provided rectangles. The two new rectangles are:

- The window position rectangle. This rectangle represents the location and size of the plug-in window. Its size is given by the intersection of the destination rectangle and the clipping rectangle. This means that the size of the window position rectangle can, and will, change frequently. Its size does not have to equal the size of the entire region being displayed. It is stored in the member variable named `m_rctWindowPos`.
- The display area rectangle. This rectangle has the same dimensions as the destination rectangle, but its origin is recalculated to be relative to the origin of the plug-in window. Because the window position rectangle is always located within the destination rectangle, the display area rectangle may have an origin comprised of negative values. It is stored in the member variable named `m_rctDisplay`.

The following diagram illustrates the relationship between the rectangles.

The sample plug-in renders by using the `StretchBlt` function to copy the bitmap to be rendered from the source rectangle to the display rectangle. Because the region to be updated (the plug-in window when rendering in windowed mode) is always contained within the display rectangle, and the display area rectangle may have an origin comprised of negative values, the operating system simply discards the portion of the bitmap that falls outside the rendering region.

You can add tracing code to your implementation of `SetRects` to print information to the Visual Studio debug window about the supplied and calculated rectangles. That way, you can observe and study this output to learn more about the behavior of the rectangles. It is particularly interesting to resize the Player during playback to
see how the rectangles change as the rendering region moves, stretches, and shrinks. You should try this with different settings in the Player for fitting the video to the Player on resize as well.

See Also

- Implementing Your Rendering Code

Implementing ProcessInput

When Windows Media Player has data to deliver to your rendering plug-in, it does so by calling IMediaObject::ProcessInput. Depending on how you made your content, this could occur infrequently or many times each second. For example, the sample rendering plug-in media contains just three lines of text that are each delivered to the plug-in at five-second intervals. An arbitrary data stream could just as easily contain full-motion video causing the Player to deliver new data constantly. It is up to the content author to specify the presentation time of each data sample.

The plug-in wizard sample code implementation of ProcessInput performs the following steps:

1. Validate the relevant parameters. The sample code checks to ensure that the provided stream index is as expected and that the IMediaBuffer interface pointer provided is not NULL.
2. Verify that the input media type has been set.
3. Show the plug-in window under certain circumstances. The plug-in creates a hidden window and makes it visible to the user at appropriate times. The following code is from the sample plug-in:

```c
if ( m_hWnd &&
    !IsWindowVisible() &&
    m_bHosted &&
    !m_bWindowless &&
    !m_bFullScreen)
{
    // Under these circumstances, be sure
    // that the plug-in window is visible.
    ShowWindow( SW_SHOW );
}
```

The preceding code first verifies that the rendering window exists. It then ensures that if the plug-in is rendering in a window hosted inside the Player user interface (and Now Playing is currently selected), and the Player isn't in full-screen mode, then the window is made visible if it is currently hidden. Your plug-in may require that the window remain hidden always, or may not use a window at all.

4. Enter a critical section. This is a simple as calling Lock. It is important that rendering plug-ins process
their data within critical sections. Otherwise, it is possible that Windows Media Player could change the data in the buffer before the plug-in is finished processing it.

5. Release any existing input buffer. The call to `ProcessInput` acts as a signal that any previous buffer processing is completed. This also means that the previous data buffer can be released by the plug-in.

6. Hold a reference on the input buffer.

7. Exit the critical section. A simple call to `Unlock` accomplishes this.

8. Invalidate the rendering region and force Windows to post a WM_PAINT message (provided that the window exists). The following code is from the `Repaint` method in the sample code:

```c
if( m_bWindowless )
{
    if( m_spWMPNodeWindowlesshost.p )
    {
        // Invalidate the RECT to which we moved the window.
        // That's the same as the region to redraw.
        hr = m_spWMPNodeWindowlesshost->InvalidateRect( &m_rctWindowPos, FALSE );
    }
}
```

The preceding code demonstrates that the plug-in must direct the Player to invalidate the rendering region when rendering in windowless mode. The following code continues the conditional block:

```c
else
{
    InvalidateRect( NULL, FALSE );
    UpdateWindow();
}
```

If rendering in windowed mode, the plug-in invalidates the rendering region itself.

**See Also**

- [Implementing Your Rendering Code](#)

When to Render

Windows Media Player rendering plug-ins that render data visually must be created to handle rendering in both windowed and windowless modes. Windowless mode is a rendering mode supported by the Windows Media Player ActiveX control where rendering occurs without a child window. This permits enhanced effects like layering or alpha-blending. When a rendering plug-in renders in windowless mode, it calls `IWMPNodeWindowlessHost::InvalidateRect` to direct the Player to invalidate the rendering region. Eventually, this results in a call to the plug-in implementation of `IWMPNodeWindowless::OnDraw`. This
method receives a handle to a device context for drawing and a pointer to a RECT structure that represents the
drawing region. OnDraw is the method where you implement your own code for drawing in windowless mode.
The following code is an excerpt from the sample rendering plug-in implementation of OnDraw:

```cpp
if ( prcDraw )
{
    HDC hDC = ( HDC )hdc;

    // Set the text color.
    COLORREF oldTextColor = ::SetTextColor( hDC, m_TextColor );
    // Set the text background color.
    COLORREF oldBkColor = ::SetBkColor( hDC, rgbWhite );

    hr = DoRendering( hDC, prcDraw );

    // Restore the original colors.
    ::SetBkColor( hDC, oldBkColor );
    ::SetTextColor( hDC, oldTextColor );
}
```

Notice that the code casts the device context handle from type OLE_HDC to type HDC. The remaining code
uses standard Windows drawing functions to prepare to draw the text data from the buffer. The text color is set
based on the user selection in the plug-in property page. The code then calls the custom function named
DoRendering to perform the actual drawing work. This is convenient because it centralizes most of the
drawing and painting code in a single function regardless of drawing mode. Finally, the code cleans-up the
changes it made earlier to the supplied device context.

When a rendering plug-in renders in windowed mode, it must invalidate the rendering region in its own window
and update the window. This results in the operating system posting a WM_PAINT message to the plug-in
window message queue. Upon receiving this message, the plug-in can paint in its own window. Because the
sample code generated by the plug-in wizard uses ATL COM, the plug-in maps the WM_PAINT message to a
function named OnPaint. This function acts as the windowed mode counterpart to
IWMPNodeWindowlessHost::OnDraw. The following code is an excerpt from the sample plug-in
implementation of OnPaint:

```cpp
PAINTSTRUCT ps;
HDC hDC;

hDC = BeginPaint( &ps );

if( NULL == hDC )
{
    return 1;
}

    // Set the text background to white.
    COLORREF oldBkColor = ::SetBkColor( hDC, rgbWhite );
    // Set the text color to the selection.
    COLORREF oldTextColor = ::SetTextColor( hDC, m_TextColor );

    hr = DoRendering(hDC, &m_rctDisplay);

    // Restore the colors.
    ::SetBkColor( hDC, oldBkColor );
    ::SetTextColor( hDC, oldTextColor );

    EndPaint( &ps );
```

The preceding code looks remarkably similar to the code from OnDraw. The main differences are:
The plug-in must create its own device context using **BeginPaint**. This is because the plug-in is painting in its window instead of a windowless region supplied by the Player.

The RECT pointer (&m_rctDisplay) the function passes to **DoRendering** is from a member variable instead of directly supplied by the Player. This RECT is calculated in **IWMPNodeRealEstate::SetRects**, which is the method that the Player calls to supply the plug-in with information about the rendering region when rendering in windowed mode. See the section titled **About SetRects**.

### See Also

- [Implementing Your Rendering Code](#)

---

**Implementing DoRendering**

**DoRendering** is the method where the sample plug-in finally displays the text data. Whether rendering in windowed mode or windowless mode, the code that ultimately displays the text data is the same. In the sample plug-in generated by the plug-in wizard, the code first creates a bitmap in memory with a white background. The code accesses the data in the input buffer from within a critical section to prevent the data from changing during this operation. The following code from **MakeBitmapFromData** illustrates this:

```c++
// Begin critical section.
Lock();

BYTE *pbInputData = NULL;
DWORD cbInputLength = 0;

// Get the data pointer and the buffer length.
if( m_spInputBuffer.p )
{
    hr = m_spInputBuffer->GetBufferAndLength( &pbInputData, &cbInputLength );
}

if ( FAILED( hr ) )
{
    Unlock();
    Return hr;
}

// Display the data from the stream.
::DrawText( m_hdcMem, ( TCHAR* ) pbInputData, cbInputLength/sizeof( TCHAR ) - 1, &m_rctSrc

// End critical section.
Unlock();
```
Once the bitmap exists, the code in **DoRendering** calls **StretchBlt** to copy the bitmap into the rendering region.

**See Also**

- [Implementing Your Rendering Code](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

**Adding Properties to the Sample Rendering Plug-in**

The rendering sample code that the Windows Media Player Plug-in Wizard generates uses a single property that represents the color of the text rendered. Your plug-in may require more than one property. You can easily add properties to your rendering plug-in in Microsoft Visual Studio using the following steps:

1. Define the methods in the interface definition code in the project's main header file. For example, to add a property called "scale", you would create two accessor methods using the following syntax:

   ```
   virtual HRESULT STDMETHODCALLTYPE get_scale(double *pVal) = 0;
   virtual HRESULT STDMETHODCALLTYPE put_scale(double newVal) = 0;
   ```

2. Add the method declarations to the main class declaration in the header file:

   ```
   STDMETHOD(get_scale)(double *pVal);
   STDMETHOD(put_scale)(double newVal);
   ```

3. Add the method implementations to the project's main CPP file:

   ```
   STDMETHODIMP CYourProject::get_scale(double *pVal)
   {
     if ( NULL == pVal )
     {
       return E_POINTER;
     }
     *pVal = m_fScale;
     return S_OK;
   }

   STDMETHODIMP CYourProject::put_scale(double newVal)
   {
     m_fScale = newVal;
     return S_OK;
   }
   ```
Finally, to make the properties accessible to the user, you'll want to make changes to the property page implementation.

See Also

- Implementing Your Rendering Code

Implementing the Property Page for a Rendering Plug-in

Windows Media Player can display a property page for each rendering plug-in to enable users to set values that change the behavior of the plug-in. Users can access the property page from the Plug-ins tab of the Options dialog box by clicking the name of the rendering plug-in to select it and then clicking Properties.

The Windows Media Player Plug-in Wizard sample code for rendering plug-ins provides a default implementation of a property page that includes three radio buttons that allow the user to select the text color. When the user clicks one of the buttons, the plug-in caches the appropriate color value in a member variable (\texttt{m\_Color}), and then enables the Apply button. When the user clicks Apply, the property page passes the color value to the plug-in object so the plug-in can change the value it uses to determine the text color during rendering.

The following sections provide more details about implementing the property page:

- About the Property Page Object
- About the Property Page Dialog Resource
- About ISpecifyPropertyPages

See Also

- Implementing Your Rendering Code

© 2000-2003 Microsoft Corporation. All rights reserved.
About the Property Page Object

The property page object is a COM object that implements the `IPropertyPage` interface. The sample code generated by the plug-in wizard uses the ATL implementation of `IPropertyPage`, which is documented in the Visual C++ documentation on MSDN. Your code at least should provide override implementations for `IPropertyPage::OnInitDialog`, which handles the event that occurs when the property page opens, and `IPropertyPage::Apply`, which handles the event that occurs when the user clicks Apply. The plug-in wizard generates sample code for each of these event handlers. The sample implementation of `OnInitDialog` retrieves a value from the registry in order to display the current text color setting. The sample implementation of `Apply` persists the value to the registry and then passes the value to the rendering plug-in property put method, named `put_color`.

Additionally, you will need to add code to handle the event that occurs when the user changes a value in a control you provide in the property page. For example, the plug-in wizard implements three functions named `OnClickedBlue`, `OnClickedRed`, and `OnClickedGreen`. When the user clicks a radio button on the property page, these are the events that cache the color value and then enable the Apply button by calling the `SetDirty` method with an argument value of TRUE.

See Also

- Implementing the Property Page for a Rendering Plug-in

About the Property Page Dialog Resource

The user interface for the property page is stored as a dialog resource. You can easily view and edit the property page dialog box in Visual Studio by selecting the ResourceView tab in the Project Workspace window, then opening the Dialog folder, and then double-clicking the property page resource name. The dialog resource editor in Visual Studio provides you with the tools you need to add controls to the property page dialog and to create event handlers that are mapped to the appropriate Windows messages. For details about how to use the resource editor in Visual Studio, refer to Visual Studio Help.

See Also

- Implementing the Property Page for a Rendering Plug-in
About ISpecifyPropertyPages

If a rendering plug-in provides a property page, the plug-in must implement the ISpecifyPropertyPages interface. This interface contains only one method: ISpecifyPropertyPages::GetPages. This is the method that associates the property page with the rendering plug-in. Windows Media Player calls this method when the user invokes the property page, passing a parameter of type CAUUID, which is a counted array of GUIDs. The sample plug-in implementation of GetPages fills this structure with a single GUID that is the class ID of the plug-in property page object. Windows Media Player then uses the class id to create the property page object.

You might notice that the sample implementation of GetPages uses CoTaskMemAlloc to allocate memory for the GUID structure. It is the responsibility of the caller, in this case Windows Media Player, to use CoTaskMemFree to release the memory. For details about the CAUUID structure, see the Platform SDK documentation.

See Also

- Implementing the Property Page for a Rendering Plug-in

Changing the Sample Rendering Plug-in Property

You will probably want to change the property that the Windows Media Plug-in Wizard creates by default. The following list details the items that might require changing:

- **The dialog resource.** Click the ResourceView tab in the Project Workspace window. Expand the folder list to open the Dialog folder. Double-click the dialog resource to open the resource editor. You can make changes to the property page dialog. For instance, you could replace the radio buttons with checkboxes.
- **The property page object code.** The default implementation uses a variable of type COLORREF to store the color value. You might require a different type of data. This would also require you to change
the code that persists the data to the registry and reads the data from the registry (including the code that reads from the registry in CProjectName::FinalConstruct).

- **The member variable that stores the property value.** This variable is named m_TextColor and is declared as type COLORREF. You may want to change the name and type of this variable throughout the project.

- **The property get and property put methods.** You might want to change the names, parameters, and implementations of these methods. Don't forget to also reflect those changes elsewhere in the project. For instance, the property page Apply method calls CProjectName::put_color.

See Also

- [Implementing Your Rendering Code](#)

### Rendering Plug-ins Programming Guide

This section describes how to modify the code generated by the Windows Media Player Plug-in Wizard to create your own Windows Media Player rendering plug-in. This section contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendering in a Separate Window</td>
<td>Describes how to modify the Plug-in Wizard sample code to create a separate window for rendering.</td>
</tr>
</tbody>
</table>

See Also

- [Windows Media Player Rendering Plug-ins](#)
Rendering in a Separate Window

The sample rendering plug-in created by the Windows Media Player Plug-in Wizard is designed to render in a window that is hosted in the Windows Media Player user interface (UI), specifically the Now Playing tab. However, you may create digital media content containing a custom stream of data that should be rendered in addition to video content. Windows Media Player does not provide a rendering region in its UI while it is rendering video, so the solution is to render the custom data in a separate window.

You can make some simple changes to the sample rendering plug-in code to enable you to render in a separate window. This section describes how to change the code to create a separate window that has a title bar, a text title, a maximize button, a minimize button, and a close button. The user can resize the window, move it to a different location on the desktop, and close the window. These are the normal behaviors that a user expects from a window; you can create your window to behave any way you choose.

This section assumes that you have already modified the sample plug-in implementation to render your content. Following the steps in this section will not create a plug-in that renders content from the sample digital media file rendering.asf provided with this SDK, because it does not contain a video stream. Do not use the sample file GUID for your media type in the code you create for this section.

Warning  Attempting to play rendering.asf using the plug-in you create in this section may yield unexpected results.

To modify the sample plug-in code, follow these steps:

1. Change the code that creates the rendering window. Set the window style to WS_OVERLAPPEDWINDOW, which provides all the features required. Also, add code to set the window title. Modify the code that creates the window in CreateRenderWin to match the following example:

   ```cpp
   // Create the plug-in window.
   HWND hWnd = Create( GetDesktopWindow(), m_rctSrc, NULL, WS_OVERLAPPEDWINDOW );
   if ( NULL == hWnd )
   {
      return E_HANDLE;
   }
   // Set the window title.
   SetWindowText( _T("Rendering Plug-in Sample") );
   ShowWindow(SW_HIDE);
   return S_OK;
   ```

2. Add code to paint the window background white when the user resizes the window. The sample plug-in depends on the StretchBlt function to fill the entire region supplied by Windows Media Player with the bitmap the plug-in creates. Your separate window will only render the bitmap at its default size, so you must repaint the window background in response to the user's actions. Add the following code to OnPaint just after the variable declarations at the beginning of the function:

   ```cpp
   // Add painting code to paint background white.
   RECT rc; // Represents the window client area.
   // Get the window RECT.
   GetClientRect( &rc );
   ```
Add the following code just after the call to `BeginPaint`:

```c
// Fill the window with a white brush.
FillRect( hDC, &rc, WHITE_BRUSH );
```

3. Change the code in `ProcessInput` that ensures the window is visible. Modify the code to match the following example:

```c
if ( m_hWnd &&
    !IsWindowVisible() &&
    !m_bFullScreen )
{
    // Under these circumstances, be sure
    // that the plug-in window is visible.
    ShowWindow( SW_SHOW );
}
```

4. Prevent the window from forwarding mouse and keyboard messages to Windows Media Player. Otherwise, the window will not respond to the user's actions. Remove the following lines of code from the ATL message map in the project's main header file:

```c
// Remove these.
MESSAGE_RANGE_HANDLER(WM_KEYDOWN, WM_KEYUP, OnPluginWindowMessage)
MESSAGE_HANDLER(WM_MOUSEACTIVATE, OnPluginWindowMessage)
MESSAGE_RANGE_HANDLER(WM_MOUSEMOVE, WM_MBUTTONDBLCLK, OnPluginWindowMessage)
MESSAGE_RANGE_HANDLER(WM_NCMOUSEMOVE, WM_NCMBUTTONDOWNCLK, OnPluginWindowMessage)
```

5. You can remove the implementations of the following interfaces.
   - IWMPNodeRealEstate
   - IWMPNodeWindowed
   - IWMPNodeWindowless

   You should remove the code in `AdviseWMPServices` that retrieves pointers to the corresponding host interfaces. You can also remove the implementation of `OnPluginWindowMessage`, which is the function that normally receives mouse and keyboard messages from the ATL message map.

See Also

- Rendering Plug-ins Programming Guide
The Microsoft Windows Media Player Software Development Kit (SDK) supports COM interfaces rendering plug-ins. The following sections document these in detail.

### Rendering Plug-ins Interfaces

The rendering plug-ins interfaces are used to connect the plug-in to Windows Media Player, exchange status information, and obtain resources from the Player. Format negotiation and data transfer are performed using the DMO interface **IMediaObject**.

The Windows Media Player SDK exposes the following interfaces for creating rendering plug-ins.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMPMediaPluginRegistrar</td>
<td>Manages plug-in registration.</td>
</tr>
<tr>
<td>IWMPNodeRealEstate</td>
<td>Obtains a rendering area from Windows Media Player.</td>
</tr>
<tr>
<td>IWMPNodeRealEstateHost</td>
<td>Requests state changes from Windows Media Player.</td>
</tr>
<tr>
<td>IWMPNodeWindowed</td>
<td>Stores a handle to the parent window used for rendering.</td>
</tr>
<tr>
<td>IWMPNodeWindowedHost</td>
<td>Sends Windows messages from the plug-in to Windows Media Player.</td>
</tr>
<tr>
<td>IWMPNodeWindowless</td>
<td>Stores a device context handle used by Windows Media Player when rendering in windowless mode.</td>
</tr>
<tr>
<td>IWMPNodeWindowlessHost</td>
<td>Provides methods to direct Windows Media Player to</td>
</tr>
</tbody>
</table>
IWMPMediaPlugin

IWMPPlugin
Manages the connection to Windows Media Player.

IWMPPluginEnable
Stores whether the plug-in is currently enabled by the user.

IWMPServices
Retrieves information from Windows Media Player about the current stream time and stream state.

IWMPWindowMessageSink
 Receives messages when in windowless mode.

See Also
- Rendering Plug-ins Programming Reference

IWMPMediaPluginRegistrar

The IWMPMediaPluginRegistrar interface manages plug-in registration.

In addition to the methods inherited from IUnknown, the IWMPMediaPluginRegistrar interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMPRegisterPlayerPlugin</td>
<td>Adds information to the registry that identifies a Windows Media Player plug-in.</td>
</tr>
<tr>
<td>WMPUnRegisterPlayerPlugin</td>
<td>Removes information about a Windows Media Player plug-in from the registry.</td>
</tr>
</tbody>
</table>

See Also
- Rendering Plug-ins Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.
IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin

The **IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin** function adds information to the registry that identifies a Windows Media Player plug-in.

**Syntax**

```c
HRESULT WMPRegisterPlayerPlugin(
    LPWSTR pwszFriendlyName,
    LPWSTR pwszDescription,
    LPWSTR pwszUninstallString,
    DWORD dwPriority,
    GUID guidPluginType,
    CLSID clsid,
    UINT cMediaTypes,
    LPVOID pMediaTypes
);
```

**Parameters**

**pwszFriendlyName**

Pointer to a wide-character null-terminated string containing the friendly name of the plug-in. This is also the name that is displayed to the user.

**pwszDescription**

Pointer to a wide-character null-terminated string containing the description of the plug-in. This information also is displayed to the user.

**pwszUninstallString**

Pointer to a wide-character null-terminated string containing the uninstall string.

**dwPriority**

Integer value containing the priority position of the plug-in in the chain of currently enabled plug-ins.

**guidPluginType**

GUID specifying plug-in type. For rendering plug-ins, specify **WMP_PLUGINTYPE_RENDERING**.

**clsid**

The class ID of the plug-in.

**cMediaTypes**
Count of media types supported by the plug-in.

*pMediaType*

Pointer to an array of media types that enumerates the supported media types. Media types are stored as type/subtype pairs.

**Return Values**

The function returns an **HRESULT**.

**Remarks**

Implement this function in the exported **DllRegisterServer** function.

The uninstall string is a command-line string that Windows Media Player passes as the argument to the Windows **ShellExecute** function when the user chooses to remove the plug-in by clicking **Remove** in the Player plug-in configuration dialog box. This gives you a way to execute your own uninstall program that initiates from Windows Media Player.

Rendering plug-ins should specify a value of 1 for **dwpriority**.

**Requirements**

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmpservices.idl; include wmpservices.h.

**Library**: Use wmp.dll.

**See Also**

- [IWMPMediaPluginRegistrar](#)
- [IWMPMediaPluginRegistrar::WMPUnRegisterPlayerPlugin](#)

© 2000-2003 Microsoft Corporation. All rights reserved.
Syntax

HRESULT WMPUnRegisterPlayerPlugin(
        GUID guidPluginType,
        CLSID clsid
    );

Parameters

guidPluginType

GUID specifying plug-in type. For rendering plug-ins, specify WMP_PLUGINTYPE_RENDERING.

clsid

Specifies the class ID of the plug-in being removed.

Return Values

The function returns an HRESULT.

Remarks

Implement this function in the exported DllUnRegisterServer function.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmpservices.idl; include wmpservices.h.

Library: Use wmp.dll.

See Also

- IWMPMediaPluginRegistrar
- IWMPMediaPluginRegistrar::WMPRegisterPlayerPlugin

© 2000-2003 Microsoft Corporation. All rights reserved.
The `IWMPNodeRealEstate` interface is implemented by the plug-in.

In addition to the methods inherited from `IUnknown`, the `IWMPNodeRealEstate` interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetDesiredSize</code></td>
<td>Returns the size of the rendering area requested by the plug-in.</td>
</tr>
<tr>
<td><code>GetFullScreen</code></td>
<td>Returns a value indicating the current rendering state of the Player.</td>
</tr>
<tr>
<td><code>GetRects</code></td>
<td>Returns the three <code>RECT</code> structures previously set in the plug-in.</td>
</tr>
<tr>
<td><code>GetWindowless</code></td>
<td>Returns a value indicating whether the plug-in is currently rendering in windowless mode.</td>
</tr>
<tr>
<td><code>SetFullScreen</code></td>
<td>Returns a value indicating whether the plug-in requests full screen rendering.</td>
</tr>
<tr>
<td><code>SetRects</code></td>
<td>Receives information about the rendering area provided by Windows Media Player.</td>
</tr>
<tr>
<td><code>SetWindowless</code></td>
<td>Receives a value indicating that the plug-in should render in windowless mode.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPNodeRealEstate::GetDesiredSize**

The `GetDesiredSize` method returns the size of the rendering area requested by the plug-in.

**Syntax**

```c
HRESULT GetDesiredSize(
    LPSIZE pSize
);
```

**Parameters**
**GetSize**

[out] A long pointer to a **SIZE** structure.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

The **SIZE** structure stores width and height as x and y coordinates. For more information about the **SIZE** structure, see the Platform SDK.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**See Also**

- **IWMPNodeRealEstate Interface**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPNodeRealEstate::GetFullScreen**

The **GetFullScreen** method returns a value indicating the current rendering state of the Player.

**Syntax**

```c
HRESULT GetFullScreen(
    BOOL* pfFullScreen
);
```

**Parameters**

- **pfFullScreen**

  [out] Pointer to a **Boolean**, true indicating full screen.

**Return Values**
The method returns an HRESULT.

Remarks

If TRUE, the Player is currently in full screen mode. If FALSE, the Player is currently displaying in full mode or skin mode.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPNodeRealEstate Interface
- IWMPNodeRealEstate::SetFullScreen

IWMPNodeRealEstate::GetRects

The GetRects method returns the three RECT structures previously set in the plug-in.

Syntax

HRESULT GetRects(
    RECT* pSrc,
    RECT* pDest,
    RECT* pClip
);

Parameters

pSrc

[out] Pointer to a RECT object that represents the portion of the original image that is to be rendered.

pDest

[out] Pointer to a RECT object that represents stretching/shrinking of the original image as well as positioning relative to the origin of the owner window.
pClip

[out] Pointer to a RECT object that represents the portion of the destination rectangle that should actually be visible to the user.

Return Values

The method returns an HRESULT.

Remarks

Any of these values can be set to NULL. The plug-in must not change values returned by this method.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPNodeRealEstate Interface
- IWMPNodeRealEstate::SetRects

IWMPNodeRealEstate::GetWindowless

The GetWindowless method returns a value indicating whether the plug-in is currently rendering in windowless mode.

Syntax

HRESULT GetWindowless(
    BOOL* pfWindowless
);

Parameters

pfWindowless

[out] Pointer to a Boolean, true indicating windowless.
Return Values

The method returns an HRESULT.

Remarks

Do not change values returned by this method.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPNodeRealEstate Interface
- IWMPNodeRealEstate::SetWindowless

IWMPNodeRealEstate::SetFullScreen

The SetFullScreen method receives a value indicating that Windows Media Player is switching to full screen mode.

Syntax

HRESULT SetFullScreen(
    BOOL fFullScreen
);

Parameters

fFullScreen

[in] Boolean, true indicating full screen.

Return Values

The method returns an HRESULT.

Remarks
Use this method to initialize any code related to full screen rendering. Do not use this method as a cue to begin rendering.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**See Also**

- [IWMPNodeRealEstate Interface](#)
- [IWMPNodeRealEstate::GetFullScreen](#)

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPNodeRealEstate::SetRects**

The **SetRects** method receives information about the rendering area provided by Windows Media Player.

**Syntax**

```cpp
HRESULT SetRects(
    const RECT* pSrc,
    const RECT* pDest,
    const RECT* pClip
);
```

**Parameters**

*pSrc*

[in] Pointer to a RECT object that represents the portion of the original image that is to be rendered.

*pDest*

[in] Pointer to a RECT object that represents stretching/shrinking of the original image as well as positioning relative to the origin of the owner window.

*pClip*

[in] Pointer to a RECT object that represents the portion of the destination rectangle that should actually be visible to the user.
Return Values

The method returns an HRESULT.

Remarks

Once the plug-in receives the RECT objects, it can immediately proceed with rendering.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPNodeRealEstate Interface
- IWMPNodeRealEstate::GetRects

IWMPNodeRealEstate::SetWindowless

The SetWindowless method receives a value indicating that the plug-in should render in windowless mode.

Syntax

HRESULT SetWindowless(
    BOOL  fWindowless
);

Parameters

fWindowless

[in] Boolean, true indicating windowless.

Return Values

The method returns an HRESULT.

Requirements
Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPNodeRealEstate Interface
- IWMPNodeRealEstate::GetWindowless

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPNodeRealEstateHost Interface

The IWMPNodeRealEstateHost interface is implemented by Windows Media Player and its methods are called by the plug-in.

In addition to the methods inherited from IUnknown, the IWMPNodeRealEstateHost interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDesiredSizeChange</td>
<td>Requests a new size.</td>
</tr>
<tr>
<td>OnFullScreenTransition</td>
<td>Requests a transition to full screen rendering.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.
IWMPNodeRealEstateHost::OnDesiredSizeChange

The **OnDesiredSizeChange** method requests a size change from Windows Media Player.

**Syntax**

```c
HRESULT OnDesiredSizeChange(
    LPSIZE    pSize
);
```

**Parameters**

*pSize*

[in] Pointer to a **SIZE** structure that represents the desired size as x,y coordinates.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

A call to this method is not a guarantee that the rendering size has actually changed. Actual rendering information is always conveyed by *IWMPNodeRealEstate::SetRects*.

For more information about the **SIZE** structure, see the Platform SDK.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**Library:** Use wmp.dll.

**See Also**

- *IWMPNodeRealEstate::SetRects*
- *IWMPNodeRealEstateHost Interface*
IWMPNodeRealEstateHost::OnFullScreenTransition

The **OnFullScreenTransition** method requests a transition to full screen rendering.

**Syntax**

```c
HRESULT OnFullScreenTransition(
    BOOL fFullScreen
);
```

**Parameters**

*fFullScreen*

[in] Boolean, true indicating full screen.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

When the Player transitions to full screen mode, it calls **IWMPNodeRealEstate::SetFullScreen**.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**Library:** Use wmp.dll.

**See Also**

- **IWMPNodeRealEstate::SetFullScreen**
- IWMPNodeRealEstateHost Interface

IWMPNodeWindowed Interface

The **IWMPNodeWindowed** interface is implemented by the plug-in.
In addition to the methods inherited from \texttt{IUnknown}, the \texttt{IWMPNodeWindowed} interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{GetOwnerWindow}</td>
<td>Returns the window handle previously set by Windows Media Player.</td>
</tr>
<tr>
<td>\texttt{SetOwnerWindow}</td>
<td>Receives a handle to the window the plug-in will use for rendering.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.

---

\textbf{IWMPNodeWindowed::GetOwnerWindow}

The \texttt{GetOwnerWindow} method returns the window handle previously set by Windows Media Player.

\textbf{Syntax}

\begin{verbatim}
HRESULT GetOwnerWindow(
    OLE_HWND* phwnd
);
\end{verbatim}

\textbf{Parameters}

\texttt{phwnd}

[out] A pointer to a window handle.

\textbf{Return Values}

The method returns an \texttt{HRESULT}.

\textbf{Remarks}

This method allows Windows Media Player to retrieve a value from the plug-in previously set by the Player. Do not change this value.

\textbf{Requirements}
**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**See Also**

- IWMPNodeWindowed Interface
- IWMPNodeWindowed::SetOwnerWindow

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**IWMPNodeWindowed::SetOwnerWindow**

The **SetOwnerWindow** method receives a handle to a parent window. The plug-in can use this parent window to derive a child window for rendering.

**Syntax**

```c
HRESULT SetOwnerWindow(
    OLE_HWND hwnd
);
```

**Parameters**

`hwnd`

[in] A handle to the window to be used for rendering.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

Rendering in a child window gives you the advantage of keeping the RECT coordinates returned from the Player consistent when the parent window position changes. Coordinate values will remain relative to the parent window, as opposed to relative to the Windows desktop.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.
IWMPNodeWindowedHost Interface

The IWMPNodeWindowedHost interface is implemented by Windows Media Player and its method is called by the plug-in.

In addition to the methods inherited from IUnknown, the IWMPNodeWindowedHost interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnWindowMessageFromRenderer</td>
<td>Sends Windows messages from the plug-in to Windows Media Player.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

IWMPNodeWindowedHost::OnWindowMessageFromRenderer

The OnWindowMessageFromRenderer method sends Windows messages from the plug-in to Windows Media Player.
Syntax

```c
HRESULT OnWindowMessageFromRenderer(
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam,
    LRESULT* plRet,
    BOOL* pfHandled
);
```

**Parameters**

* uMsg

* wParam

* lParam

* plRet
  [out] Pointer to a long integer containing the return value.

* pfHandled
  [out] Pointer to a Boolean, true indicating handled.

**Return Values**

The method returns an HRESULT.

**Remarks**

You must forward certain messages received by your plug-in window to Windows Media Player if they are not handled by your window code. For instance, keyboard messages and mouse messages might need to be handled by the Player or Internet Explorer. Do not forward messages received from Windows Media Player in IWMPWindowMessageSink::OnWindowMessage.

The following window messages must be forwarded:

- WM_KEYDOWN
- WM_KEYUP
- WM_LBUTTONDOWN
- WM_LBUTTONUP
- WM_MBUTTONDOWN
- WM_MBUTTONUP
- WM_MBUTTONDBLCLK
 IWMPNodeWindowless Interface

The IWMPNodeWindowless interface is implemented by the plug-in.

In addition to the methods inherited from IWMPWindowMessageSink, the IWMPNodeWindowless interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDraw</td>
<td>Receives a handle to a device context and a RECT</td>
</tr>
</tbody>
</table>
The `OnDraw` method receives a handle to a device context and a `RECT` structure for drawing in windowless mode.

Syntax

```c
HRESULT OnDraw(
    OLE_HDC  hdc,
    const RECT*  prcDraw
);```

Parameters

- `hdc`
  - [in] A handle to a device context.

- `prcDraw`
  - [in] A `RECT` structure representing the drawing area.

Return Values

The method returns an `HRESULT`.

Remarks

This is the method that initiates rendering when in windowless mode.

Requirements

**Version:** Windows Media Player 9 Series or later.
**IWMPNodeWindowlessHost Interface**

The **IWMPNodeWindowlessHost** interface is implemented by Windows Media Player and its method is called by the plug-in.

In addition to the methods inherited from **IUnknown**, the **IWMPNodeWindowlessHost** interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InvalidateRect</strong></td>
<td>Requests that Windows Media Player refresh the display area.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

---

© 2000-2003 Microsoft Corporation. All rights reserved.
Syntax

HRESULT InvalidateRect(
    const RECT* prc,
    BOOL fErase
);

Parameters

prc  
[in]  Specifies the region to update.

fErase
[in]  Flag, true indicating erase.

Return Values

The method returns an HRESULT.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

Library: Use wmp.dll.

See Also

- IWMPNodeWindowlessHost Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPPlugin Interface

The **IWMPPlugin** interface is implemented by the plug-in. It manages the connection to Windows Media Player.

In addition to the methods inherited from **IUnknown**, the **IWMPPlugin** interface exposes the following methods.
**Method**

**AdviseWMPServices**

**Description**

Receives a pointer to a Windows Media Player interface that contains methods that provide stream state information.

**GetCaps**

Sets a value that specifies whether the plug-in requires the input format and output format to be identical.

**GetID**

Returns the class ID of the plug-in.

**Init**

Receives a playback context identifier.

**Shutdown**

Executes when Windows Media Player shuts down the plug-in.

**UnAdviseWMPServices**

Executes when Windows Media Player releases the pointer provided in **AdviseWMPServices**.

**See Also**

- Rendering Plug-ins Interfaces

---

**Previous**

**Next**

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Windows Media 9 Series**

**Previous**

**Next**

---

**IWMPP::AdviseWMPServices**

The **IWMPP::AdviseWMPServices** method is implemented by the plug-in.

**Syntax**

```c++
HRESULT AdviseWMPServices(
    IWMPServices* pWMPServices
);
```
Parameters

$pWMPServices$

[in] Pointer to an $IWMPServices$ interface.

Return Values

The method returns an $HRESULT$.

Remarks

Windows Media Player calls the $AdviseWMPServices$ method on the plug-in to pass in a pointer that the plug-in can then use to call the $IWMPServices$ interface, which contains methods that provide information about the current state of the stream.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- $IWMPPlugin$ Interface
- $IWMPServices$ Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

IWMPPlugin::GetCaps

The $IWMPPlugin::GetCaps$ method returns a flag that specifies whether the plug-in can convert between an input format and an output format.

Syntax

```
HRESULT GetCaps(
    DWORD* pdwFlags
);
```

Parameters

$pdwFlags$
[out] Pointer to a variable that specifies whether the plug-in can convert formats. The specified value is a bitwise combination of zero or more flags from the WMPPlugin_Caps enumeration.

**Return Values**

The method returns an HRESULT.

**Remarks**

There are currently two possible [out] values that the plug-in may specify: zero to indicate that the plug-in can convert formats, or WMPPlugin_Caps_CannotConvertFormats, which forces Windows Media Player to handle any necessary format conversion.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**See Also**

- IWMPPlugin Interface
- WMPPlugin_Caps

---

IWMPPlugin::GetID

The IWMPPlugin::GetID method returns the class id of the plug-in.

**Syntax**

```c
HRESULT GetID(
    GUID* pGUID
);
```

**Parameters**

- `pGUID`

  [out] Pointer to a GUID that represents the class ID of the plug-in.

**Return Values**
The method returns an HRESULT.

Remarks

For more information on the GUID structure, see the Platform SDK.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPPPlugin Interface

IWMPPlugin::Init

The IWMPPlugin::Init method is called when Windows Media Player initializes the plug-in.

Syntax

HRESULT Init(
    DWORD dwPlaybackContext
);

Parameters

dwPlaybackContext

[in] DWORD value that indicates the particular Windows Media Player playback engine to which the plug-in belongs.

Return Values

The method returns an HRESULT.

Remarks

It is possible at any given time that multiple instances of Windows Media Player could be running in the same process. For instance, multiple Windows Media Player control instances could be embedded in the same
browser window, or even in multiple instances of a browser that co-exist in the same process. It is also possible that the same instance of Windows Media Player could create multiple playback engines at the same time. The \textit{dwPlaybackContext} value allows you to determine which instance of the Windows Media Player playback engine contains the plug-in. This is useful if you wish to enable multiple plug-ins to connect to each other.

\textbf{Init} and \textbf{Shutdown} will always be called on the same thread.

\textbf{Requirements}

\textbf{Version}: Windows Media Player 9 Series or later.

\textbf{Header}: Defined in wmprealestate.idl; include wmprealestate.h.

\textbf{See Also}

- \textbf{IWMPPPlugin Interface}

\textbf{IWMPPlugin::Shutdown}

The \textbf{IWMPPlugin::Shutdown} method is called when Windows Media Player shuts down the plug-in.

\textbf{Syntax}

\begin{verbatim}
HRESULT Shutdown();
\end{verbatim}

\textbf{Return Values}

The method returns an \textbf{HRESULT}.

\textbf{Requirements}

\textbf{Version}: Windows Media Player 9 Series or later.

\textbf{Header}: Defined in wmprealestate.idl; include wmprealestate.h.

\textbf{Remarks}

\textbf{Init} and \textbf{Shutdown} will always be called on the same thread.

\textbf{See Also}
The **IWMPPlugin::UnAdviseWMPServices** method is used to release the pointer provided by **AdviseWMPServices**.

### Syntax

```c
HRESULT UnAdviseWMPServices();
```

### Parameters

The method takes no parameters.

### Return Values

The method returns an **HRESULT**.

### Remarks

Windows Media Player calls this method when the pointer provided by **AdviseWMPServices** is no longer valid. The plug-in should use this method to cease making stream state requests through the pointer.

### Requirements

**Version**: Windows Media Player 9 Series or later.

**Header**: Defined in wmprealestate.idl; include wmprealestate.h.

### See Also

- **IWMPPlugin Interface**
- **IWMPPlugin::AdviseWMPServices**
The **IWMPPPluginEnable** interface sets and retrieves a value that represents whether the plug-in has been enabled by the user.

In addition to the methods inherited from **IUnknown**, the **IWMPPPluginEnable** interface exposes the following methods.

### Method Description

- **GetEnable**
  - Retrieves the current enable state.
- **SetEnable**
  - Sets the current enable state.

**See Also**

- Rendering Plug-ins Interfaces

---

**IWMPPPluginEnable::GetEnable**

The **IWMPPPluginEnable::GetEnable** method returns a value indicating whether the user has enabled the plug-in.

**Syntax**

```cpp
HRESULT GetEnable(
    BOOL* pfEnable
);
```

**Parameters**

- `pfEnable`
Pointer to a Boolean value indicating whether the user has enabled the plug-in.

Return Values

The method returns an HRESULT. For rendering plug-ins, return E_NOTIMPL.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealstate.idl; include wmprealstate.h.

See Also

- IWMPPPluginEnable Interface
- IWMPPPluginEnable::SetEnable

IWMPPluginEnable::SetEnable

The IWMPPluginEnable::SetEnable method retrieves a value indicating whether user has enabled the plug-in.

Syntax

HRESULT SetEnable(
    BOOL fEnable
);

Parameters

fEnable

[in] A variable that receives a value indicating whether the user has enabled the plug-in.

Return Values

The method returns an HRESULT. For rendering plug-ins, return E_NOTIMPL.

Requirements

Version: Windows Media Player 9 Series or later.
 IWMPServices Interface

The IWMPServices interface is implemented by Windows Media Player. It provides methods to retrieve the current stream state and current stream time.

In addition to the methods inherited from IUnknown, the IWMPServices interface exposes the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetStreamState</td>
<td>Returns a value that represents the current stream state.</td>
</tr>
<tr>
<td>GetStreamTime</td>
<td>Returns a value that indicates the current stream time.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

© 2000-2003 Microsoft Corporation. All rights reserved.
The **IWMPServices::GetStreamState** method retrieves information about the current play state of the stream.

**Syntax**

```cpp
HRESULT GetStreamState(
    WMPServices_StreamState* pState
);
```

**Parameters**

`pState`

[in] A pointer to a **WMPServices_StreamState** enumeration value.

**Return Values**

The method returns an **HRESULT**.

**Remarks**

The stream is stopped, paused, or playing.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmprealestate.idl; include wmprealestate.h.

**Library:** Use wmp.dll.

**See Also**

- **IWMPServices Interface**
- **WMPServices_StreamState**

© 2000-2003 Microsoft Corporation. All rights reserved.

---

The **IWMPServices::GetStreamTime** method retrieves a structure indicating the current stream time.

**Syntax**

The **IWMPServices::GetStreamTime** method retrieves a structure indicating the current stream time.
HRESULT GetStreamTime(
    REFERENCE_TIME* ptt)
);

Parameters

ptt       [in]  Pointer to a REFERENCE_TIME structure.

Return Values

The method returns an HRESULT.

Remarks

The current stream time is determined by Windows Media Player. This means that the value returned by this method do not necessarily represent the elapsed time relative to the beginning of the file. For instance, if the user moves the trackbar in the Player to seek the media to a new position, the value returned by this method returns the time elapsed since the media began playing from the new position. Changes in playback rate will also affect the value returned by this method.

The values provided in the rtTimestamp member of IMediaObject::ProcessInput and the rtTimestamp member of the DMO_OUTPUT_DATA_BUFFER structure supplied by IMediaObject::ProcessOutput contain values that indicate when the data provided in the buffer will be rendered relative to the current stream time. Therefore, these values also do not necessarily represent the elapsed time relative to the beginning of the file or the presentation time specified in the file.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

Library: Use wmp.dll.

See Also

- IWMPServices Interface
IWMPWindowMessageSink Interface

The IWMPWindowMessageSink interface receives messages when the rendering plug-in does not have a window hosted in the Player UI.

In addition to the methods inherited from IUnknown, the IWMPWindowMessageSink interface exposes the following method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnWindowMessage</td>
<td>Receives window messages.</td>
</tr>
</tbody>
</table>

See Also

- Rendering Plug-ins Interfaces

IWMPWindowMessageSink::OnWindowMessage

The OnWindowMessage method receives window messages.

Syntax

```c++
HRESULT OnWindowMessage(
    UINT uMsg,
    WPARAM wparam,
    LPARAM lParam,
    LRESULT* plRet,
    BOOL* pfHandled
);
```

Parameters

- `uMsg`
- `wparam`
lpParam


plRet

[out] Pointer to a long integer containing the return value.

pfHandled

[out] Pointer to a Boolean, true indicating handled.

Return Values

The method returns an HRESULT.

Remarks

When a rendering plug-in window is not hosted in the Windows Media Player UI, such as when the Player is in windowless mode or the Player is rendering Windows Media video, it needs a mechanism to receive window messages from the Player. In these cases, the Player forwards window messages to the plug-in by calling this method.

Windows Media Player queries all active rendering plug-ins to determine whether they support this method and automatically forwards window messages to each plug-in that does.

Requirements

Version: Windows Media Player 9 Series or later.

Header: Defined in wmprealestate.idl; include wmprealestate.h.

See Also

- IWMPWindowMessageSink Interface

© 2000-2003 Microsoft Corporation. All rights reserved.

Rendering Plug-ins Enumeration Types

The Windows Media Player SDK implements the following enumeration types for creating Rendering plug-ins.
**WMPPlugin_Caps**

The **WMPPlugin_Caps** enumeration type signals whether the plug-in can convert between input and output formats.

**Syntax**

```c
enum WMPPlugin_Caps{
    WMPPlugin_Caps_CannotConvertFormats = 1
};
```

**Members**

**WMPPlugin_Caps_CannotConvertFormats**

The plug-in requires that the input format and output format be the same.

**Remarks**

When **IWMPPPlugin::GetCaps** returns **WMPPlugin_Caps_CannotConvertFormats**, Windows Media Player handles any necessary format conversion.

**Requirements**

**Version:** Windows Media Player 9 Series or later.

**Header:** Defined in wmpservices.idl; include wmpservices.h.
The `WMPServices_StreamState` enumeration indicates whether the stream is currently stopped, paused, or playing.

### Syntax

```cpp
eenum WMPServices_StreamState{
    WMPServices_StreamState_Stop = 0,
    WMPServices_StreamState_Pause = 1,
    WMPServices_StreamState_Play = 2
} WMPServices_StreamState;
```

### Members

- **WMPServices_StreamState_Stop**
  
  The stream is stopped.

- **WMPServices_StreamState_Pause**
  
  The stream is paused.

- **WMPServices_StreamState_Play**
  
  The stream is playing.

### Requirements

- **Version**: Windows Media Player 9 Series or later.
- **Header**: Defined in wmpservices.idl; include wmpservices.h.

See Also

- Rendering Plug-ins Enumeration Types
- IWMPPlugin::GetCaps
Windows Media Metafiles

This reference documents Windows Media metafiles, which use the .wax, .wvx, .wmx, and .asx file name extensions. It contains overview and programming guide sections, and a full reference section on metafile element tags, their attributes and values, and special conditions related to each element.

A metafile is a file that contains information about other files. A metafile can be used to list a group of media content files that are to be played in order. Windows Media metafile playlists, simply referred to as playlists in this reference document, are one of the most powerful features of Microsoft Windows Media Technologies. Playlists allow you to control and customize your media content. For instance, with playlists you can schedule content to play in succession or insert advertising or special interest clips into a presentation. A further advantage of playlists is that instead of playing a stream, stopping, starting the next stream, and then waiting for it to finish buffering, Windows Media Services and Windows Media Player work together to play the clips one after the other with minimal buffering time or interruption between them.

Windows Media Technologies and other Internet products provide you with the tools to understand user demographics and dynamically customize a broadcast or message for individual users.

This reference is divided into the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Windows Media Metafiles</td>
<td>Introduces Windows Media metafile elements and discusses their purpose.</td>
</tr>
<tr>
<td>Windows Media Metafile Guide</td>
<td>Details the steps necessary for creating metafiles. Examples illustrate how to use element tags for specific tasks.</td>
</tr>
<tr>
<td>Windows Media Metafile Reference</td>
<td>Explains in detail each of the metafile elements, their attributes and values, and special conditions related to each. Explains metafile file name extensions and their proper use.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player 9 Series SDK
About Windows Media Metafiles

Microsoft Windows Media metafiles are text files that provide information about a file stream and its presentation. Windows Media metafiles are based on the Extensible Markup Language (XML) syntax and can be encoded in ANSI or UNICODE (UTF-8) format. They are made up of various elements with their associated tags and attributes. Each element in a Windows Media metafile defines a particular setting or action in Windows Media Player. Some elements must be located in a specific position in the file relative to other elements. Some elements have required tags and attributes that must be defined in the Windows Media metafile.

Windows Media Download (WMD) packages provide a way to combine Windows Media Player borders, playlist information, and multimedia content into a single downloadable file.

The following sections provide conceptual information about Windows Media metafile playlists and Windows Media Download packages.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Media Metafiles Overview</td>
<td>Describes Windows Media metafiles, why you need them, and how to use them.</td>
</tr>
<tr>
<td>Windows Media Download Packages</td>
<td>Describes WMD packages, why you need them, and how to use them.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Metafiles
Windows Media Metafiles Overview

The most important part of successfully using Windows Media metafiles is using the correct syntax for the metafile elements. Syntax errors in a Windows Media metafile can cause anything from a single attribute being overlooked, to the metafile not being recognized as valid and failing to work.

Almost as important is the order in which the elements appear in a Windows Media metafile. The attributes of some elements temporarily override the attributes of similar elements in different sections of the metafile. There is a defined Order of Precedence.

Windows Media metafile playlists are Windows Media metafiles that provide information that Windows Media Player uses to receive unicast streams, multicast streams, and other supported media from an intranet or the Internet. A metafile playlist is basically a shortcut to media content. A metafile playlist can be sent as e-mail, used as a link reference on a Web page, created dynamically using Active Server Pages (ASP), or exist as a stand-alone file on a local disk drive. A metafile playlist can reference another metafile playlist, an ASP page, or a Windows Media Station file (with a .nsc file name extension). An .nsc file is used to define a Windows Media Station to Windows Media Player. The basic handling process is the same for each case.

See Also

- About Windows Media Metafiles

Purpose of a Metafile Playlist

The basic purpose of a metafile playlist is to redirect streaming media content away from browsers to Windows Media Player because most browsers will attempt to download the content instead of stream it. A metafile playlist provides information Windows Media Player uses to receive streams, such as the Uniform Resource Locator (URL). The metafile playlist can provide the path to media files over the Internet, an intranet or just stored media on a network or local drive. With processing redirected to Windows Media Player, you have greater control over the streaming media.

Metafile playlists are used to control the streaming experience. By combining different media files into a single content stream, you can control and customize their presentation.

See Also

- Windows Media Metafiles Overview
Using a Metafile Playlist

Because you cannot link directly from a Web page to a streaming media server, you use metafile playlists. The metafile playlists contain the information needed by Windows Media Player and are stored on a Web server. You can then link from the Web document to a metafile playlist. When a metafile playlist is opened, control transfers to Windows Media Player, which processes the file, connects to the streaming media server, and plays the specified content.

The browser first downloads the metafile playlist to the user's cache directory. Because the metafile playlist is small, this is a very quick step. The user's computer then finds in its file associations table the association of the metafile playlist with Windows Media Player. Windows Media Player opens and interprets the scripting in the metafile playlist, which contains, among other things, the URL of the streaming content. Windows Media Player uses the URL to locate the content and initiate the stream. The metafile playlist scripting then controls the streaming experience.

Because metafile playlists work through a helper application associated with the ASX MIME type (application/mplayer2 or video/x-ms-asf), they are compatible with any browser that supports helper applications. The examples shown in this document will work with Microsoft Internet Explorer 4.0 and later, and with Netscape Navigator 4.0 and later on Microsoft Win32® and Apple Macintosh platforms. In all examples, you will have to be sure that any media files referenced have valid paths and file names for your environment.

See Also

- Windows Media Metafile Guide
- Windows Media Metafile Reference
- Windows Media Metafiles Overview
Support for Multiple Languages

Windows Media Player 9 Series supports Windows Media metafiles created using the Unicode character set. This allows you to include multilingual metadata in your metafile playlist. The following rules govern the use of multilingual metadata in Windows Media metafiles:

- Characters must be encoded using the UTF-8 encoding scheme.
- The metafile playlist must include the following `PARAM` at the playlist level:

  `<PARAM NAME = "Encoding" VALUE = "utf-8">`

- Only Windows Media Player 9 Series or later supports this functionality.
- If the metafile playlist is not saved with UTF-8 encoding and the correct `PARAM` element, it will be parsed using the default system locale code page of the user's computer.

See Also

- `PARAM` Element
- Windows Media Metafiles Overview

Windows Media Download Packages

Windows Media Download (WMD) packages combine Windows Media Player borders, playlist information, and multimedia content in a single downloadable file with a .wmd file name extension. A WMD package could include an entire album of music videos that also displays advertising in the form of graphical branding and links to an online music retailer Web site.

Users can download a WMD package from a Web site simply by clicking a link. Once the package is downloaded to the user's computer, Windows Media Player extracts the files inside the package automatically, then adds the packaged playlist to the playlists drop-down box, adds the content to Media Library, displays the border skin in the Now Playing pane of the full mode Windows Media Player, and then plays the initial item in the playlist.

WMD packages provide the following benefits:

- A single-click downloads, extracts, and catalogs multiple files to the user's computer.
- Content plays immediately after being downloaded.
- Customized borders can display advertising and branding information.
- Packaged playlists are cataloged in Media Library.
Web site redirection can occur from within Windows Media Player to a related site. A variety of interactive border controls are available. Multiple audio and video file types are supported for packaging.

All of these benefits can be used together. For example, a single WMD package could give users the opportunity to view the lyrics to songs as they play, display a video that accompanies the songs, view advertising information about the record distributor, view album cover art, visit a fan Web site, and catalog the content in Media Library.

The following sections provide concepts to help you understand and create Windows Media Download packages.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Windows Media Download Packages Work</td>
<td>Provides an overview of how a WMD file is packaged, posted to a Web site, downloaded, and played by Windows Media Player.</td>
</tr>
<tr>
<td>Using Borders in Windows Media Download Packages</td>
<td>Introduces borders and explains how a border is created.</td>
</tr>
<tr>
<td>Using Playlists in Windows Media Download Packages</td>
<td>Describes how a metafile playlist file is used in a WMD package, and provides sample code.</td>
</tr>
<tr>
<td>Creating a Windows Media Download Package</td>
<td>Describes the process of putting a package together for distribution.</td>
</tr>
</tbody>
</table>

See Also

- About Windows Media Metafiles
- Borders for Windows Media Player

© 2000-2003 Microsoft Corporation. All rights reserved.

How Windows Media Download Packages Work

A Windows Media Download package is launched from a Web site when a user clicks a link in a Web browser, such as Microsoft Internet Explorer. This action opens Windows Media Player and then downloads and un-packages the WMD package on the user's hard disk in a default folder.

Once the files have been extracted from the WMD package, Windows Media Player locates a Windows Media metafile playlist with a .asx file name extension among the packaged files. If it finds one, the Player creates a
playlist based on the included metafile. Files that contain multimedia content are then added to **Media Library**.

Windows Media Player also looks for a **SKIN** element in the metafile. If the **SKIN** element contains a reference to a border file with a .wmz file name extension, the Player loads the border into the **Now Playing** pane. The Player then starts to play the content provided in the package.

The following diagram shows how content is packaged in a WMD file, posted to a Web site, downloaded, and played on a client computer using Windows Media Player.

The following table describes the three elements that make up a WMD package.

<table>
<thead>
<tr>
<th>Package element</th>
<th>Function</th>
<th>File name extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Border</td>
<td>A fixed, customized user interface created by the content owner for displaying, linking, and playing all media packaged in the WMD package. The techniques used to create borders are similar to those used to create skins.</td>
<td>.wmz</td>
</tr>
<tr>
<td>Metafile Playlist</td>
<td>A Windows Media metafile that contains <strong>ENTRY</strong> elements, playlist information, and a <strong>SKIN</strong> element identity for content files.</td>
<td>.asx</td>
</tr>
<tr>
<td>Multimedia Content</td>
<td>A file containing any audio or video format that is supported by Windows Media Player.</td>
<td>.wma, .wmv, .ASF, .wav, .avi, .mpg, .mp3</td>
</tr>
</tbody>
</table>

**See Also**

- [Windows Media Download Packages](#)
Using Borders in Windows Media Download Packages

Borders enable you to create a customized graphical user interface for your packaged content. The border can include elements such as images, interactive controls, and links to Web sites. You can use borders in cases where you want to add additional value to your packaged content, such as for branding or advertising. After users download and open your WMD package, Windows Media Player automatically displays your custom border when it plays the packaged content.

Unlike a skin, which enables users to completely replace the Windows Media Player user interface, a border is displayed only in the Now Playing pane of the full mode Player. However, the same tools and technologies that you use to create skins are also used to create borders. The following illustration shows a border.
It is important to understand the basic techniques for creating a skin before attempting to create a border. Border programming is accomplished using two programming languages: Extensible Markup Language (XML) and Microsoft JScript. XML is used to define interface elements such as buttons, sliders, and text boxes. You don't need to understand all the details of XML since you don't have to write new XML code elements; you can simply use the ones provided by Windows Media Player. Although JScript is not required for creating borders, it can be used to provide additional functionality.

A compressed border file with a .wmz file name extension includes a border definition file with a .wms file name extension and all the image files used within the border.

To include a border in a WMD package, simply create a border and reference that border in a Windows Media metafile playlist. The border file is loaded into Windows Media Player after the Player parses the metafile and interprets the `SKIN` element that references the border. The `SKIN` element is used only for borders, and the `HREF` attribute of the `SKIN` element can reference only one skin for each package.

See Also

- Borders for Windows Media Player
- Windows Media Download Packages
- Windows Media Player Skins
Using Playlists in Windows Media Download Packages

Playlists are Windows Media metafiles with a .asx file name extension that provide information that tells Windows Media Player how to play the packaged content. By combining multiple content files into a single WMD package, you can control and customize your Windows Media Download package using the playlist.

Note In general, metafile playlists are used by WMD packages to reference the multimedia content in the package, and not a stream from a server on an intranet or the Internet. However, URL references within the .asx file are supported.

Using XML, the metafile provides the information Windows Media Player uses to play and display content. Playlists are made up of various XML code elements with their associated tags and attributes. Each element in a Windows Media metafile playlist defines a particular setting or action in Windows Media Player.

The user's computer associates a Windows Media metafile that has an .asx file name extension with Windows Media Player. Windows Media Player opens and parses the XML code in the metafile, which contains the path for locating the packaged audio or video files. The metafile script then controls the audio, video, and graphical experience. The metafile also contains information that Windows Media Player processes and displays in the playlist drop-down box. Immediately after the list is displayed, the first item in the list is played.

A metafile playlist is a shortcut to the files that contain your packaged content. The following code is an example of a metafile that specifies the border to display by using the SKIN element, two songs to be played, and the playlist information for each song.

```
<ASX Version="3.0">
<AUTHOR>Name of content creator goes here</AUTHOR>
<TITLE>Album Title goes here</TITLE>
<PARAM name="Album" value="Album Title ">
<PARAM name="Artist" value="Artist Name" />
<PARAM name="Genre" value="Genre" />
<Skin HREF="myborder.wmz"/>
<Entry>
<Ref HREF="song1.wma"/>
<AUTHOR>Creator's name</AUTHOR>
<COPYRIGHT>Copyright information</COPYRIGHT>
<TITLE>Song #1 title</TITLE>
</Entry>
<Entry>
<Ref HREF="song2.wma"/>
<AUTHOR>Creator's name</AUTHOR>
<COPYRIGHT>Copyright information</COPYRIGHT>
<TITLE>Song #2 name</TITLE>
</Entry>
</ASX>
```
Creating a Windows Media Download Package

Follow these steps to create a WMD package.

1. **Create a border.** Use the same techniques you would use to build a skin for Windows Media Player. Design the border so that resizing Windows Media Player will not ruin the composition of the border elements. For instance, use a solid color or visualization as a background because these will scale well as the Player is resized.

2. **Compress the border contents.** Using a compression program commonly used for distributing and storing files, compress the border files: images, JScript files, and the skin definition file with a .wms file name extension. Rename the compressed file so that it has a .wmz file name extension. Windows Media Player associates and reads the .wmz file extension automatically.

3. **Write a Windows Media metafile.** The Player will not load the border unless you create a Windows Media metafile with an .asx file name extension that implements the **SKIN** element. The metafile can also be used to create a playlist that describes the content included in the package.

4. **Assemble your content.** Put all the files that you want to use into a folder. This includes audio files, video files, metafiles, and border definition files.

5. **Create the package.** Using a compression program commonly used for distributing and storing files, compress the border file, content files, and the metafile into a new file with a .wmd file name extension. Windows Media Player associates and reads the .wmd file extension automatically.

6. **Post the package to a Web site.** The completed package is ready to be posted to a Web site and downloaded by users.

**See Also**

- [Borders for Windows Media Player](#)
- [Windows Media Download Packages](#)
- [Windows Media Metafiles](#)
Windows Media Metafile Guide

A Windows Media metafile can be as simple or complex as you need it to be. The most basic Windows Media metafile contains only the Uniform Resource Locator (URL) of some multimedia content on a server. The client, Windows Media Player, parses this information and then opens the media file or stream defined in the Windows Media metafile. A complex metafile can contain multiple files or streams arranged in a playlist, instructions on how to play the files or streams, text and graphic elements such as title, author, and copyright text, personalized ad insertion into a live stream, hyperlinks associated with elements on the Windows Media Player interface, and more.

The following sections provide detailed information on how to create and use Windows Media metafile playlists.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of Playlists</td>
<td>Lists available file name extensions.</td>
</tr>
<tr>
<td>Creating Metafile Playlists</td>
<td>Describes how to create Windows Media metafile playlists.</td>
</tr>
<tr>
<td>Metafile Playlists</td>
<td>Describes metafile playlist usage, scripting, metadata, and processing.</td>
</tr>
<tr>
<td>Metafile Extension Guidelines</td>
<td>Describes the preferred use of file name extensions for streaming media files.</td>
</tr>
<tr>
<td>Order of Precedence</td>
<td>Describes how metafile playlist elements override other metafile playlist elements.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafiles

© 2000-2003 Microsoft Corporation. All rights reserved.
Types of Playlists

Windows Media metafile playlists, simply referred to as playlists in this document, have .wax, .wvx, or .asx extensions. Playlists with these extensions are used to access Windows Media files with .wma, .wmv, and .asf file name extensions, respectively.

See Also

- Metafile Extension Guidelines
- Using Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

Creating Metafile Playlists

You can create a playlist using any text editor, such as Microsoft Notepad. Open your text editor. Type the script entries you want to implement. After you have finished typing into Notepad, save the file with an appropriate file name and file name extension. For more information about extensions, see Metafile Extension Guidelines. Typically the file name is the name of the Windows Media file or stream followed by an extension of .wax, .wvx, or .asx. For example, if your media content is a Windows Media audio file that has a .wma extension, use the .wax extension when naming the playlist. Playlists must not include any formatting codes from a word processor, such as Microsoft Word. To be sure no formatting codes are included in the playlist, save the file as a plain text or ASCII file.

Note Elements and attributes are not case sensitive. The text used in the playlist to define an element or attribute can be either uppercase or lowercase, or a mixture of both.

If an element does not have any child elements (those that modify or are contained within another element), a single slash character (/) can be used at the end of the opening tag, just before the '>', in place of a closing tag. Child elements of an element must appear between the opening and closing tag for that element, otherwise they are not child elements for that element, and are ignored or cause an error in the syntax of the playlist.

The first four characters of a playlist must be "<ASX". The ASX element is used in all playlists whether their extension is .wax, .wvx, or .asx. There must be only one ASX element per playlist. This element identifies the file as a Windows Media metafile playlist. It does not specify the type of playlist.

The ASX element has three possible attributes:

VERSION
The **VERSION** attribute is required and must follow immediately after the **ASX** element, for example "<ASX version = "3.0">". The current version number is 3.0. Windows Media Player supports all previous versions. Acceptable values for the **VERSION** attribute include both 3.0 and 3 (with no decimal point).

### PREVIEWMODE

The **PREVIEWMODE** attribute is optional. It provides another mechanism for specifying how long to render a clip. If the value of the **PREVIEWMODE** attribute is YES, Windows Media Player will render each clip for the duration specified by the element **PREVIEWDURATION**. Each clip can have a **PREVIEWDURATION** specified.

### BANNERBAR

The optional **BANNERBAR** attribute defines whether the Windows Media Player control reserves space for a banner graphic. (Use the **BANNER** element to specify the graphic to display.) If the value of **BANNERBAR** is FIXED, Windows Media Player reserves banner space for the show and for every clip, whether or not the metafile playlist specifies a banner for the show or clip. This will keep the size of the Windows Media Player window the same (except when the video size changes) regardless of the absence or presence of a banner graphic. If the show or clip does not have a banner associated with it, the space reserved for one is black. If the value of the **BANNERBAR** attribute is AUTO, Windows Media Player reserves space for the banner only when the show or clip includes one.

```xml
<ASX version="3.0" BANNERBAR="AUTO" >

For more information about the three attributes of the **ASX** element, see the reference entry for the **ASX Element**.

An **ASX** element contains **ENTRY** child elements that define information about the media files to be accessed. Each **ENTRY** element must contain a **REF** element that specifies the path to the media file to be streamed. There must be at least one **ENTRY** or **ENTRYREF** element within an **ASX** element.

Other elements defined within the scope of the **ASX** element, such as **TITLE** and **AUTHOR**, are associated with the metadata displayed by Windows Media Player.

The simplest playlists are created by adding multiple **ENTRY** elements with a single **REF** element to a metafile. Each **ENTRY** element in a metafile playlist is rendered in the order it appears in the file as though the user had manually opened each clip.

**Example Code**

```xml
<ASX version = "3.0">
<!--A simple playlist with entries to be played in sequence.-->
  <Title>The Show Title</Title>
  <Entry>
    <Ref href = "mms://adventure-works.com/Path/title1.wma" />
  </Entry>
  <Entry>
    <Ref href = "mms://adventure-works.com/Path/title2.wma" />
  </Entry>
  <Entry>
    <Ref href = "mms://adventure-works.com/Path/title3.wma" />
  </Entry>
</ASX>

Be sure that the playlist is working by double-clicking it in Windows Explorer. Windows Media Player should
open and start streaming the media content. After you have confirmed that the playlist works, save it to your Web server along with your Web pages, and link to it by means of an HREF element, or embed it in a Web page using the Windows Media Player OBJECT element.

The following sections contain more information:

- Nesting Metafiles
- Using ASP Pages to Dynamically Create Windows Media Metafile Playlists

See Also

- BANNER Element
- Example Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

Nesting Metafiles

A metafile can reference another metafile. To reference another metafile, use the ENTRYREF element. An ENTRYREF element in the primary metafile points to an external metafile.

The Windows Media Player control processes the ENTRY elements of the referenced metafile as if they were included in the primary metafile at the position of the ENTRYREF element. However, it skips any ENTRY elements in the referenced metafile that have the SKIPIFREF attribute set to YES.

The Windows Media Player 7.0, 7.1, and Windows Media Player for Windows XP controls ignore any ENTRYREF elements in the referenced metafile. Thus, metafiles can only be nested one level deep using these versions. These versions also ignore the ASX element and its attributes in the referenced file. Windows Media Player 9 Series or later supports nesting metafiles up to 5 deep.

See Also

- Creating Metafile Playlists
Using ASP Pages to Dynamically Create Windows Media Metafile Playlists

You can use Active Server Pages (ASP, or .asp files) to dynamically generate playlists based on information provided by users. An ASP page is a dynamic Web page used in conjunction with Microsoft Internet Information Services (IIS). ASP is an environment in which you can combine HTML, scripts, and reusable ActiveX server components to create dynamic and powerful Web-based business solutions. ASP pages enable server-side scripting for IIS with native support for both Microsoft Visual Basic Scripting Edition (VBScript) and Microsoft JScript. This discussion assumes that you are familiar with ASP and defining variables.

All header information must be contained on the first line of the ASP page string returned to Windows Media Player.

When you use ASP pages to generate playlists, you must specify values for the Response object's ContentType and expires properties in the ASP page because of latency issues with Windows Media Player. The Response.ContentType value must be a valid file name extension for Windows Media metafiles. Acceptable values include wma, wax, wmv, wvx, asf, and asx.

The Response.expires property specifies the length of time, in seconds, that Windows Media Player caches the playlist file. Specifying a value of zero results in Windows Media Player requesting a new playlist from the server each time the user refreshes the page.

See the Platform SDK for details about using the Response object in Active Server Pages.

The following code is an example of an ASP page used to generate a Windows Media metafile playlist.

```<%Response.ContentType = "video/x-ms-wma"%><%Response.expires=0 %>
<ASX VERSION="3.0">
  <TITLE>Your title here</TITLE>
  <ENTRY>
    <REF HREF ="mms://adventure-works.com/pubpt/filename.wma" />
  </ENTRY>
</ASX>
```

See Also

- Creating Metafile Playlists
Metafile Playlists

Windows Media metafile playlists are used to organize stream content and presentation. A playlist can simply redirect Windows Media Player to the content to be streamed, but more commonly will provide a list of files to be played sequentially. A more complex playlist may contain repeats, commercial inserts, and information about the items in the list.

Another advantage of playlists is that instead of playing a stream, stopping, starting the next stream, and then waiting for it to finish buffering, Windows Media Services and Windows Media Player work together to play the clips one after the other with minimal buffering time or interruption.

The following sections provide more information about metafile playlists:

- Control of Playlists
- Custom Graphics in Windows Media Player
- Displaying Web Pages in Windows Media Player
- Logging Stream Data
- Retrieving Metadata

See Also

- Windows Media Metafile Guide

Control of Playlists

Control of playlists is maintained by the client. Playlists are parsed by Windows Media Player, which takes actions based upon the elements of the playlist. The client has control of the streaming experience.

See Also

- Metafile Playlists
Custom Graphics in Windows Media Player

With Windows Media metafile scripting, you can add graphics elements to Windows Media Player as the stream plays. There is one basic type of graphics you can add.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANNER</td>
<td>Using the <strong>BANNER</strong> element you can place a banner (32 pixels high by 194 pixels wide) image on Windows Media Player at the bottom of the video display area. You can then use a child element to add a hyperlink to the banner.</td>
</tr>
</tbody>
</table>

See Also

- Metafile Playlists
- Modifying the Display
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

Displaying Web Pages in Windows Media Player

Windows Media metafiles let you display Web pages while the entries in your playlist play. These Web pages display in the **Now Playing** pane in place of the visualization or the normal display of a video file. Using regular HTML, you can supplement your digital media with custom content or advertisements, or provide a custom user interface to supplement the user interface of the Player.

To specify HTML content to embed in the Player, use the **PARAM** element and set its **NAME** attribute to
"HTMLView" and its \texttt{VALUE} attribute to a URL. If the \texttt{PARAM} element is a child of the \texttt{ASX} element, it displays for the entire duration of a playlist. If it is a child of an \texttt{ENTRY} element, it displays for the duration of that entry only. This lets you specify supplemental content that will change depending on what is playing.

\textbf{Note} The URL you specify cannot refer to a file on the local machine. This prevents metafiles in the local security zone from loading Web pages which might contain malicious script code.

The following example illustrates a playlist that contains two entries, each of which references a different URL:

\begin{verbatim}
<ASX version="3.0">
  <TITLE>HTMLView Demo</TITLE>
  <ENTRY>
    <TITLE>First Entry</TITLE>
    <REF href="audio.wma"/>
    <PARAM name="HTMLView" value="http://www.microsoft.com"/>
  </ENTRY>
  <ENTRY>
    <TITLE>Second Entry</TITLE>
    <REF href="video.wmv"/>
    <PARAM name="HTMLView" value="http://www.proseware.com"/>
  </ENTRY>
</ASX>
\end{verbatim}

\textbf{Displaying Video}

While HTML is showing in the \textbf{Now Playing} pane, video files cannot display as they normally do. Instead, you can display them within the Web page by using an embedded Windows Media Player control. This allows you to specify the exact dimensions of the video display and to coordinate its appearance with the other HTML content.

When embedding the control in this way, do not specify the \texttt{URL} attribute of the \texttt{OBJECT} element that embeds the control. The digital media content referenced by the playlist \texttt{ENTRY} element plays within the embedded control automatically.

You can manipulate the embedded control using JScript code as if it were a standalone Web page. Because the embedded control recognizes that it is contained within the Player, it automatically communicates with the Player when JScript code runs. This means you can create a custom user interface to supplement or replace the user interface provided by the Player itself. A custom \textbf{Stop} button provided in the HTML, for example, can duplicate the functionality of the normal \textbf{Stop} button of the Player.

For general information about embedding the Windows Media Player control in a Web page, see the Player Control Guide.

\textbf{Additional Tips}

When you design Web pages that will be shown in the Now Playing area of the Player, take into account the reduced screen space that you have available. In general, it's not a good idea to simply embed Web pages that were originally designed for a standalone browser. Such pages will often be clipped on one or more sides and end users will have to use scroll bars to see all of your visual content.

You also cannot specify the size of the display window. The elements of your Web page should maintain a logical appearance when the user resizes the Player window.

Be aware that script commands may not work as you intend. The \texttt{FILE} and \texttt{URL} script commands are affected by the end user's security choices. The Web page that you display in the Player is subject to the Player security
choice, and the default setting is to not execute those two script commands. Custom script commands will always execute, however, so you can take similar actions through custom commands.

See Also

- Metafile Playlists

Logging Stream Data

Logged information can be acquired and used to determine viewer behavior, for example, how often a stream is viewed, or if a specific user viewed a stream and for how long at what quality.

Logging information is automatically sent to the server from which the playlist originated. You can also send logging information to additional servers, including Web servers you use exclusively for logging. To do this, use the LOGURL element, specifying a valid URL for the HREF attribute. You can include LOGURL elements as children of the ASX element and as children of individual ENTRY elements. When the playlist is first opened, logging information is sent to the origin server and to each URL specified in LOGURL children of the ASX element. Then, as each entry is reached, logging information specific to that entry is sent to each URL specified in LOGURL children of the ENTRY element.

The Windows Media Format SDK supports the LOGURL element through the IWMSReaderNetworkConfig interface and the following methods:

```c
HRESULT AddLoggingUrl(LPCWSTR pwszUrl);
HRESULT GetLoggingUrl(DWORD dwIndex, LPCWSTR pwszUrl, DWORD *pcchUrl);
HRESULT GetLoggingUrlCount(DWORD *pdwUrlCount);
HRESULT ResetLoggingUrlList();
```

In addition to the information that is automatically logged, a metafile playlist can log custom information through the use of the PARAM element. To use the PARAM element in this way, set the NAME attribute to "log:" followed by a log field name and an optional XML namespace separated from the field name by another colon ("::"). Everything after the second colon is treated as a namespace, so the field name should not contain a colon.

The log field specified in the NAME attribute is set to the value of the VALUE attribute. If the log does not already contain a field with the specified name, it will be added.

Example Code

```xml
<ASX version="3.0">
</ASX>
```
Retrieving Metadata

While a show or clip is playing, your script can retrieve metadata, such as title and author, by using the `getItemInfo` methods of the Windows Media Player `Media` and `Playlist` objects. You can retrieve metadata from the ASX scope using `Playlist` object methods and from the ENTRY scope using `Media` object methods.

For example, to retrieve the values for AUTHOR, ABSTRACT and PARAM in the file below, use the `getItemInfo` method of the Playlist object. The attribute name is required for this method. Attribute names can be obtained by providing the index number to the `attributeName` property. The available indexes for a Playlist object can be obtained using the `attributeCount` property.

**Example Code**

```xml
<ASX version="3.0">
  <AUTHOR>My Talking File</AUTHOR>
  <ABSTRACT>Talking File Album</ABSTRACT>
  <PARAM name="one" value="111"/>
  <ENTRY>
    <REF href="Artists_Only.wma"/>
    <TITLE>Artists Only</TITLE>
    <COPYRIGHT>2000</COPYRIGHT>
    <PARAM name="three" value="333"/>
  </ENTRY>
  <PARAM name="two" value="222"/>
</ASX>
```
To retrieve the values of the current Media object in the ENTRY scope for REF, TITLE, COPYRIGHT, and PARAM ("three"), use the currentMedia property of the Player object. Use the attributeCount property of the Media object to determine the number of attributes available for the specified Media object. Use index numbers with the getItemInfoByAtom method to retrieve attribute values. Use index numbers with the getAttributeName method of the Media object to determine the names of the available attributes, and then use the results with the getItemInfo method.

For an example of using Windows Media Player object methods to retrieve metadata, see Playlist.attributeCount.

See Also

- Creating Metafile Playlists
- Metafile Playlists
- Windows Media Metafile Guide

Using Metafile Playlists

Playlists specify how the streaming media or media files will be played and what metadata Windows Media Player will display.

This section explains several ways you can use playlists. The section is divided into the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifying the Display</td>
<td>Adding text, links, and images.</td>
</tr>
<tr>
<td>Redirection</td>
<td>Using playlists to redirect the browser to Windows Media Player and specifying the URL of a stream or media file to play.</td>
</tr>
<tr>
<td>Accessing Media</td>
<td>Using metafile elements and their child elements to specify the content to access, and the order and duration of their playback.</td>
</tr>
<tr>
<td>Using Live Event Stream Switching</td>
<td>Using the EVENT element to specify a media stream to access and then resume playing the original stream. Used for ad insertion.</td>
</tr>
<tr>
<td>Using Metafiles for Seamless Stream Switching</td>
<td>Using the EVENT element to preload the next media stream to access to avoid gaps in the presentation.</td>
</tr>
</tbody>
</table>
Modifying the Display

Playlists can alter the Windows Media Player user interface in four main ways:

- Text properties
- Image properties
- MOREINFO properties
- ABSTRACT text

Text properties

Windows Media Player enables the display of Title, Author, Copyright, and Description metadata text. Clip metadata can come from the stream or media file, or it can come from a playlist. Show metadata comes from the playlist. In general, playlists are a better method of passing text properties to Windows Media Player, especially if text elements are likely to change. It is easier to edit text in a playlist than to re-author a media file. And because properties read from a playlist override those contained in the media file, you can easily update display text by adding the new text to the corresponding property in a playlist. In the following example, the text of the Title and Author metadata in the playlist overrides the Title and Author text contained in the media file, sample.wma.
**DESCRIPTION** text is retrieved from a Windows Media file referenced in an **ENTRY** element unless there is an **ABSTRACT** element in a metafile playlist. If there is **ABSTRACT** text, it will be displayed, overriding the **DESCRIPTION** text.

```xml
<ASX version="3.0" BANNERBAR="AUTO" >
  <ENTRY>
    <BANNER HREF="YourPath\2.gif">
      <TITLE>Upgrade</TITLE>
      <AUTHOR>Ad Department</AUTHOR>
      <REF href="YourPath\sample.wma"/>
    </BANNER>
  </ENTRY>
</ASX>
```

**Image properties**

Banner images can be added to the user interface of Windows Media Player. The graphic can be used for advertising, providing information, and providing access to Web sites, to name a few possibilities.

Use the **BANNER** element to specify a graphic image (32 pixels high by 194 pixels wide) for display by Windows Media Player. The graphic is displayed below any video content. A hyperlink can be added to the banner using the **MOREINFO** child element.

A ToolTip can be defined by an **ABSTRACT** element within the scope of the **BANNER** element. Any defined ToolTip text can be displayed by pausing the mouse pointer over the banner graphic. Selecting the banner graphic with the mouse pointer will activate any hyperlink defined with the **MOREINFO** element.

The preferred **BANNER** graphics format is the GIF format. The JPG format can be used if the graphic is properly sized.

The previous example illustrates use of the **BANNER** element.

**Note** **BANNER** images are not supported with DRM files or when Windows Media Player is embedded in a Web page.

For more information about banners, see [Custom Graphics in Windows Media Player](#).

**MOREINFO properties**

Text and image areas of the user interface can be associated with URLs. During playback, users can select one of these sections to connect to the URL associated with it in their Web browser. For example, you can associate an advertiser's Web site with an ad banner image as shown in the following code snippet.

```xml
<BANNER HREF="YourPath\2.gif">
  <ABSTRACT>More Information.</ABSTRACT>
  <MOREINFO HREF="http://www.proseware.com" />
</BANNER>
```

**ABSTRACT text**

**ABSTRACT** text is used to display a short pop-up description of the text or image areas of the user interface it is associated with. During playback, if the mouse pointer hovers over one of these areas, a ToolTip appears beside the mouse pointer displaying the **ABSTRACT** text associated with the area. **ABSTRACT** text is retrieved from a metafile and is defined with the **ABSTRACT** element. The **ABSTRACT** element can be a child element of either an **ENTRY** or a **BANNER** element.
Redirection

The playlist will redirect the browser to Windows Media Player to play the designated media stream or media file. The most basic metafile playlist contains only the Uniform Resource Locator (URL) of a media file.

To create a basic metafile playlist, open your favorite text editor, such as Microsoft Notepad, and type the following example code.

```xml
<ASX version="3.0">
  <ENTRY>
    <REF HREF="PathToYourFile"/>
  </ENTRY>
</ASX>
```

Substitute a valid path to your Windows Media file using the syntax in the following table.

<table>
<thead>
<tr>
<th>Source of content</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content is a file on a Windows Media server</td>
<td>mms://ServerName/Path/FileName.wma</td>
</tr>
<tr>
<td>Content is a broadcast multicast that is accessed from a Windows Media station</td>
<td><a href="http://WebServerName/Stations/kxyz.nsc">http://WebServerName/Stations/kxyz.nsc</a></td>
</tr>
<tr>
<td>Content is a broadcast unicast that is accessed from a publishing point on a Windows Media server</td>
<td>mms://ServerName/PublishingPointAlias</td>
</tr>
<tr>
<td>Content is a file on a Web server</td>
<td><a href="http://WebServerName/Path/Filename.wma">http://WebServerName/Path/Filename.wma</a></td>
</tr>
<tr>
<td>Content is a file on a local hard disk</td>
<td>file://c:\Path\Filename.wma</td>
</tr>
<tr>
<td>Content is a file on a network share</td>
<td>file://\ServerName\Path\Filename.wma</td>
</tr>
<tr>
<td>Content is a file on a Windows Media server</td>
<td>mms://ServerName/Path/FileName.wmv</td>
</tr>
</tbody>
</table>
Save the file you have created with a file name and extension as described in File Name Extensions. Double-click it in Windows Explorer. Windows Media Player should open and start streaming the content. You can save the file to your Web server, along with your Web pages, and link to it with an HREF element, or embed it in a Web page using the Windows Media Player OBJECT tag.

See Also

- Metafile Playlists
- Using Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

© 2000-2003 Microsoft Corporation. All rights reserved.

Accessing Media

Use playlists to specify and to control the streaming media or media files that Windows Media Player plays.

Use the ENTRY element to specify a single media element (a media file or a live stream) and any child elements (such as images, MOREINFO links, and ABSTRACT text). Use an ENTRYREF element to specify a playlist. A playlist can contain one or more ENTRY or ENTRYREF elements. Windows Media Player executes a playlist by starting with the first entry and then playing each entry in turn until the list is finished.

An ENTRY element can point to any type of media that Windows Media Player can play. This includes not only .wma, .wmv, .asf, and .avi files, to name a few, but live streams as well. By using a series of ENTRY or ENTRYREF elements to reference media content, you can use a playlist to send a single stream that consists of multiple sources. The referenced streams will play sequentially and be seen as one continuous stream by the viewer. For example, the playlist can contain two ENTRY elements: a standard introduction from a Windows Media file with a .wma extension, and a live Windows Media stream.

Note A playlist must not contain links to media files that have content created with different versions of Digital Rights Management (DRM). In a metafile playlist, if there are links for media files with DRM version 1 content and for media files created with later DRM versions, Windows Media Player will only play the DRM version 1 content.

Controlling Playback

Use playlists to control not only which media clip is played, but also which portions of the clip are played and how. You can use playlists to define a set of clips to be looped or repeated, to set the duration of play, and to assign start times, and start and end markers for each entry. The STARTTIME, STARTMARKER, and ENDMARKER elements work in conjunction with markers in the media file.
For example, the following playlist uses an ad banner and the associated MOREINFO link in one ENTRY, and references a STARTMARKER and ENDMARKER.

```xml
<ASX version ="3.0">
  <Title>Windows Media Example</Title>
  <Abstract>Windows Media Technologies</Abstract>
  <Entry>
    <Title>This is the first Entry --></Title>
    <Author>Ad Department</Author>
    <Copyright>2000</Copyright>
    <Abstract>This is a description of the ad.</Abstract>
    <Ref href="ad.wma"/>
    <Banner href ="purchase.gif">
      <Abstract>Click here to go to our Web site.</Abstract>
    </Banner>
  </Entry>
  <!-- This is the second Entry -->
  <Entry>
    <Title>Playlist Clip Number Two</Title>
    <Author>Windows Media</Author>
    <Copyright>2000</Copyright>
    <Ref href="show.wma"/>
    <StartMarker Name = "Segment2" />
    <EndMarker Name = "Segment3" />
  </Entry>
</ASX>
```

### Setting Duration

Use the DURATION element to specify how long to play a clip or set of clips. You can also use the PREVIEWMODE attribute of the ASX element in conjunction with the PREVIEWDURATION element to specify how long to play a clip or set of clips. Set the PREVIEWMODE attribute to YES to use the PREVIEWDURATION element to specify how long to play the associated clip. The PREVIEWDURATION and DURATION elements have the same behavior.

### See Also

- Metafile Playlists
- Using Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide
Using Live Event Stream Switching

Streaming media can also be controlled by the interaction of script commands embedded in a media stream with Windows Media metafile elements in a metafile playlist.

An event is a particular type of script command embedded in a media stream or media file. When the Windows Media Player control receives the script command, it processes the event as defined by the EVENT element in the metafile playlist. Windows Media Player switches from the current stream it is rendering and renders the content referenced in the metafile playlist EVENT element. The EVENT element is usually used in live production.

An EVENT element looks similar to an ENTRY element, but each handles the playback of streams and media files differently. The ENTRY element is used to create playlists. A stream or media file referenced in an ENTRY element starts playing as soon as the stream or media file referenced in the previous ENTRY finishes. A stream referenced in an EVENT plays only when a specific script command is received. For example, when Windows Media Player receives a script command with type string "EVENT" and the command string "Adlink", it searches the playlist for the following elements.

```xml
<Event NAME="Adlink" WHENDONE="RESUME">
  <Entry HREF=mms://www.proseware.com/adlink.wma />
</Event>
```

Windows Media Player then switches from the live stream to play the stream or media file contained in the EVENT, in this case Adlink.wma. The code WHENDONE="RESUME" instructs Windows Media Player to resume playing the previous stream as soon as Adlink.wma is finished.

**Note** Failure to handle every event embedded in a media stream or media file may yield unexpected results.

If you want to use live event stream switching, you must include one EVENT element in your playlist to handle each event script command embedded in the media streams or media files in your playlist. Before you create your playlist, you must know the details about which script commands are embedded in your digital media content. If there is an event script command that you want Windows Media Player to ignore, include an EVENT element in your playlist to handle the event, but reference a dummy URL in the event handler.

**Ad Insertion**

This technique can be used for ad insertion. For example, during a live Internet broadcast of a ball game, a command can be sent at the beginning of every commercial break that instructs each client (Windows Media Player) to play commercials listed in its playlist. When clients finish playing the commercials, the playlist instructs each client to cut back to the live broadcast. The EVENT media content will be rendered only when the streaming media being accessed broadcasts embedded scripting with the matching EVENT name.

The possibilities inherent in EVENT switching are best appreciated by contrasting how ads reach viewers through standard, over-the-air broadcasting with how ads can reach viewers using Windows Media Technologies. Historically, broadcast ads could only be roughly targeted at viewers, using ratings data as the primary criteria. Ads sent using Windows Media Technologies can be aimed directly at the target user because EVENTS and playlists can be built on the fly based on user input. For more information, see Personalizing Media Delivery.

You can also use metafile playlists to display customized graphics, audio and text for advertising. You can use the BANNER element as a child element of an EVENT to display an advertising message graphic. The
**BANNER** element provides the path and file containing the graphics for your advertising banner. You can also provide a link to a site or file using the **MOREINFO** child element. The URL in the **MOREINFO** element can provide a link to even more advertisements on the Web.

**Example Code**

```xml
<BANNER HREF="SomePath\2.gif">
  <ABSTRACT>Read This Ad and Buy.</ABSTRACT>
  <MOREINFO HREF="http://www.proseware.com" />
</BANNER>
```

The following example inserts the ad Advert.wma into the broadcast unicast stream BallGame when a client receives a script command **EVENT** with the **NAME** attribute set to "Time-Out". **CLIENTSKIP** is set to NO to prevent the streamed ad from being skipped. In this example, the streamed ad must be played before returning to the original stream. When the ad is finished, the client resumes playing the original stream.

**Example Code**

```xml
<ASX VERSION="3.0">
  <ENTRY>
    <REF HREF="mms://proseware.com/BallGame" />
  </ENTRY>
  <EVENT NAME="Time-Out" WHENDONE="RESUME">
    <ENTRY>
      <REF HREF = "mms://proseware.com/Advert.wma"
           CLIENTSKIP = "NO" />
    </ENTRY>
  </EVENT>
</ASX>
```

**See Also**

- Metafile Playlists
- Using Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

© 2000-2003 Microsoft Corporation. All rights reserved.
for Windows Media Player is to open the next media stream referenced by the playlist 20 seconds before the end of the currently rendered stream. This generally provides a seamless transition between media streams, depending on other factors such as Web access times.

Use the `EVENT` element in a playlist in conjunction with `OPENEVENT` commands from the encoder to facilitate seamless switching between streams or files. Sending an `OPENEVENT` command 20 seconds or more prior to the `EVENT` command can minimize delays in stream switching. Then Windows Media Player is able to preload a portion of the upcoming streaming content into a buffer.

Use Windows Media Encoder to send a script command in the stream using the following format:

```
OPENEVENT eventname
```

The event name must be the one defined in the `EVENT` element in the playlist. When Windows Media Player receives an `OPENEVENT` script command from the encoder, it looks to the `EVENT` element in the playlist and begins buffering the clip or stream defined in the `EVENT` element. Windows Media Player then holds this information until the actual event of the same name. When the named event is received, Windows Media Player switches to that previously buffered content.

**Note** You cannot use Unicode characters for the `OPENEVENT` script command in the media file or the `EVENT` element in the playlist.

**See Also**

- [Creating Metafile Playlists](#)
- [Metafile Playlists](#)
- [Player.ScriptCommand Event](#)
- [Using Metafile Playlists](#)
- [Windows Media Metafile Elements Reference](#)
- [Windows Media Metafile Guide](#)

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**Using Announcements**

An announcement is a file that contains information about the URL for a media stream, including the multicast IP address, port, stream format, and other station settings. Announcements are created by Windows Media Administrator when a unicast or multicast publishing stream is created. The client can quickly load the announcement file, then proceed to access the streaming media file.

For a unicast publishing point, the publishing point media stream is opened. For a multicast publishing point, the URL is extracted from a broadcast station file with an .nsc extension, and the streaming media is accessed. Unlike a unicast stream, no header information is contained in a multicast stream. That information comes from
the broadcast station file with an .nsc extension. Windows Media Player usually first opens an announcement file, which is one use for metafile playlists, that points to the location of the broadcast station file.

Example Code

```xml
<ASX VERSION="3.0">
  <TITLE>title</TITLE>
  <ENTRY>
    <REF HREF="mms://proseware.com/pubpoint"/>
  </ENTRY>
</ASX>
```

See Also

- Creating Metafile Playlists
- Using Metafile Playlists

Using URL and Server Rollover

You can use metafile playlists to provide a means of automatically rolling over to alternate content sources when a stream cannot be accessed or played. You can use this rollover method to specify sources of the same content on different servers or different types of servers. You can, for example, specify a first alternate on a different Windows Media server. If that content fails to play, the client can roll over to a second alternate on a Web server. Windows Media Player automatically tries to roll over to different protocols according to its Windows Media property settings before trying the rollover URLs in the playlist.

Set server and protocol rollover by placing multiple `REF` elements in succession within one `ENTRY` element. Each `REF` element specifies an alternate location or protocol for a media file or stream.

Example Code

```xml
<!--Server and protocol rollover is set for the file Rollover.wma.-->
<ASX version="3.0">
  <TITLE>MyServer Rollover</TITLE>
  <ENTRY>
    <REF HREF="mms://Server1.proseware.com/Path/Rollover.wma"/>
    <REF HREF="mms://Server2.proseware.com/Path/Rollover.wma"/>
    <REF HREF="mms://Server3.proseware.com/Path/Rollover.wma"/>
    <REF HREF="http://www.proseware.com/Path/Rollover.wma"/>
  </ENTRY>
</ASX>
```

See Also
Using Custom Parameters and Commands

You can create custom parameters to pass additional metadata in a metafile playlist by using the `PARAM` element. Use the `NAME` attribute of the `PARAM` element to define the custom parameter name. Use the `VALUE` attribute to define the value for the named custom parameter.

Retrieve `PARAM` metadata by using the `getItemInfo` methods of the `Media` and `Playlist` objects. For an example using these methods to retrieve metadata, see the `attributeCount` method of the `Playlist` object.

The following example metafile playlist shows the use of the `PARAM` element to define custom parameters.

```xml
<ASX version="3.0" BANNERBAR="auto" >
  <TITLE>Example Media Player Show</TITLE>
  <PARAM NAME="Director" VALUE="Jane D." />
  <ENTRY>
    <TITLE>Example Clip</TITLE>
    <REF HREF="http://www.proseware.com/media.wma" />
    <PARAM NAME="Title" VALUE="Example Clip" />
    <PARAM NAME="Location" VALUE="North America" />
    <PARAM NAME="Release Date" VALUE="March 1998" />
  </ENTRY>
  <ENTRY>
    <TITLE>Another Clip</TITLE>
    <REF HREF="http://www.proseware.com/more_media.wma" />
    <PARAM NAME="Title" VALUE="Another Clip" />
    <PARAM NAME="Location" VALUE="Japan" />
    <PARAM NAME="Release Date" VALUE="December 2000" />
  </ENTRY>
</ASX>
```

See Also

- [Using Metafile Playlists](#)
Personalizing Media Delivery

Unlike one-way communication that broadcasts identical content to a mass audience, Windows Media Technologies provides you with the tools to use demographics to individualize broadcasts. With the Internet, two-way communication on a large scale is readily available. This dynamic interchange of information enables content providers to know their audience and respond with customized presentations.

The following example, using a fictional company, illustrates how delivery of streaming content can be personalized. This discussion assumes that you are familiar with Active Server Pages (ASP, or .asp files) and defining variables.

News Network is a fictional broadcast news organization that has expanded its operations to include a Web site. The main feature of the site is a section where users can create their own personalized newscasts. Instead of viewing a traditional newscast that is aimed at a mass audience, a user views a complete news program that contains only topics of personal interest. The following sequence describes a typical user experience.

1. A new user goes to the site, and clicks Create Your Personal Newscast.
2. A preference form opens. On this form, the user answers questions regarding personal preferences, such as favorite news stories, least favorite news stories, hobbies, and usual method of receiving daily news.
3. The user sends this information, and a few seconds later views a complete, 15-minute, personal newscast containing program content, transitions, and commercials. Selection of each media element, including commercials, is based on the user profile, and is accomplished automatically with Windows Media Technologies components and off-the-shelf Internet tools.

The following list describes how the various tools interact to create a personalized newscast.

1. The preference form that the user fills out is an Active Server Page (ASP) (Choices.asp). Data obtained from the preference form is analyzed by two server components. One component uses the information to query a Microsoft SQL Server database of news stories. The other component is an ad server that uses a complex set of rules based on contractual requirements and demographics to schedule ads appropriate to the user at that time.
2. The two databases return different portions of a playlist. The news story database returns a set of appropriate story Entries, and the ad server returns a set of appropriate commercial Entries.
3. A second ASP page (PlayShow.asp) receives the Entries from the news story database and ad server, and combines those with standard show open, close, and transition Entries. All Entries are then laid out according to the template provided by PlayShow.asp, and the ASP page returns a playlist to the user.
4. The embedded Windows Media Player control on the user's computer plays the playlist from beginning to end, and the user views a personalized newscast.

The following example is a portion of a playlist file that a user might receive. Ad banners, MOREINFO links, and ABSTRACTS have been added to it.

<ASX Version="3">
<TITLE>MyPersonalized NewsCast</TITLE>
<ENTRY ClientSkip="no">
<!—- Commercial Element 1 -->
<REF HREF="mms://proseware.com/Commercial.wma" />
The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.

See Also

- [Creating Metafile Playlists](#)
- [Metafile Playlists](#)
- [Using Metafile Playlists](#)
- [Windows Media Metafile Elements Reference](#)
- [Windows Media Metafile Guide](#)
Relative links are a fully supported feature of Windows Media metafiles. You can use relative links in metafiles much like you use them in HTML documents. The use of relative links enables you to create metafiles that are portable, meaning you can copy or move an entire directory structure to another server without updating the paths to graphic files used as banners or the HREF attributes of MOREINFO elements (if they reference files on the same Web server as the stored metafile). Relative links work, in any application that supports them, because the parts of the URL not included in the HREF attribute of an element are included in the URL sent by the application to the server when that URL is requested. This means that the protocol (such as http://), the server name, and the virtual directory in which the file containing the relative link is located are all included in the URL that is sent to the server. If the media file, or URL you link to using a relative link does not reside on the same server as the metafile, the relative link is not valid.

Note This information on relative links is correct only when using a link to metafiles that are opened in the stand-alone Windows Media Player. When you use the embedded Windows Media Player control, all links are relative to the page on which the control is hosted, unless the BASE child element of the ASX element is used to redirect the relative path.

All relative links in the metafile must be fully relative, not drive-relative, for streaming media. When a URL begins with a "\" character, the link is drive-relative. Windows Media Player attempts to open the file linked to on the drive where the metafile was opened from, usually a Web server. Because Web servers use virtual directories, Windows Media Player tries to find the specified stream or media file in a subdirectory of a virtual directory on the Web server where the metafile was opened from. A user would not have access to a server directory. The example in this section illustrates the use of a fully relative link.

You can use drive-relative links when using metafiles on a single computer where all media files linked to in the metafile playlist exist on a storage device in that computer. However, you cannot stream media in this manner.

When you use a link to another metafile to allow for relative links, the name displayed as the Title by Windows Media Player is the Title in the original metafile.

Relative links cannot be used for URLs to a streaming media server because different protocols are used for accessing streaming media content.

In the following example playlist, the first ENTRYREF contains a URL for another playlist, relative.wax.

```xml
<ASX>
  <ENTRYREF HREF="http://example.microsoft.com/metafiles/relative.wax"/>
</ASX>
```

The following example is the file relative.wax, which contains relative links.

```xml
<ASX version = "3.0">
  <BANNER HREF = "graphics/logo1.jpg">
    <MOREINFO HREF = "category1/category1.htm" />
  </BANNER>
  <ENTRY>
    <REF HREF = "mms://samples.microsoft.com/myfile.wma" />
  </ENTRY>
</ASX>
```

The URL containing the reference to the playlist, relative.wax, can exist in a metafile playlist anywhere that is accessible to the user. For the URL to be processed successfully, there must be a Web server named "example.microsoft.com". That Web server must have a virtual directory defined as "metafiles". The referenced playlist, relative.wax, must exist on the Web server named "example.microsoft.com" in the virtual directory "metafiles".
For the referenced media files in the playlist relative.wax to be successfully accessed and played, there must be a directory named "Graphics" which is a subdirectory of the server's virtual directory "metafiles". The graphics file Logo1.jpg, referenced in the BANNER element, must be the subdirectory named Graphics.

Similarly a document named Category1.htm must reside in a subdirectory named Category1. The directory named Category1 must exist as a subdirectory of the virtual directory "metafiles" on the Web server example.microsoft.com.

See Also

- Creating Metafile Playlists
- Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

Example Playlists

Following are three example playlists. To use these examples you must ensure that you have substituted valid paths and file names that are accessible to Windows Media Player.

<table>
<thead>
<tr>
<th>Example playlist</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Basic Playlist</td>
<td>A simple playlist.</td>
</tr>
<tr>
<td>An Advanced Playlist</td>
<td>A playlist with a more complete set of playlist elements.</td>
</tr>
<tr>
<td>An Example Radio Station Playlist</td>
<td>A playlist that scans three radio stations.</td>
</tr>
</tbody>
</table>

See Also

- Creating Metafile Playlists
- Metafile Playlists
A Basic Playlist

The following playlist example shows a basic set of playlist elements. When writing your own code, you will need to change all URLs and file names to valid file names that are accessible to your Windows Media Player.

Code Example

<ASX version = "3.0">
<TITLE>Basic Playlist Demo</TITLE>
<ENTRY>
<TITLE>An Entry in a basic playlist</TITLE>
<AUTHOR>Microsoft Corporation</AUTHOR>
<COPYRIGHT>(c)2000 Microsoft Corporation</COPYRIGHT>
<REF HREF = "mms://proseware.com/path/Yourfile.wma" />
</ENTRY>
</ASX>

The following table provides details on the use of each element in the example code.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ASX version = &quot;3.0&quot;&gt;</td>
<td>The <strong>ASX</strong> element identifies for the client (Windows Media Player) that this is a Windows Media metafile playlist. The <strong>version</strong> attribute specifies the version number of the metafile elements.</td>
</tr>
<tr>
<td>&lt;TITLE&gt;Basic Playlist Demo&lt;/TITLE&gt;</td>
<td>The <strong>TITLE</strong> element identifies the title of the playlist as a whole. Windows Media Player displays this metadata as the show title.</td>
</tr>
<tr>
<td>&lt;ENTRY&gt;</td>
<td>Begins the <strong>ENTRY</strong> element. An <strong>ENTRY</strong> element is a way to define a particular clip in a playlist.</td>
</tr>
<tr>
<td>&lt;TITLE&gt;An Entry in a the basic playlist&lt;/TITLE&gt;</td>
<td>Identifies the title of the playlist clip. It is different than the previous <strong>TITLE</strong> element because this one is nested within an <strong>ENTRY</strong> element. Windows Media Player displays this metadata as the clip title.</td>
</tr>
<tr>
<td>&lt;AUTHOR&gt;Microsoft Corporation&lt;/AUTHOR&gt;</td>
<td>The <strong>AUTHOR</strong> element identifies the author of the media clip. Windows Media Player displays this metadata as the clip <strong>AUTHOR</strong>.</td>
</tr>
<tr>
<td>&lt;COPYRIGHT&gt;(c)1999 Microsoft Corporation&lt;/COPYRIGHT&gt;</td>
<td>The <strong>COPYRIGHT</strong> element identifies any copyright associated with the media content. This information is displayed as the clip <strong>COPYRIGHT</strong> by Windows Media Player.</td>
</tr>
<tr>
<td>&lt;!-- This is a comment. Change the following path to point to your Windows media file --&gt;</td>
<td>A comment. <strong>Comments</strong> are only be visible when the code is viewed, and are in the same format as <strong>XML</strong> comments.</td>
</tr>
</tbody>
</table>
<REF HREF = "mms://proseware.com/path/Yourfile.wma" />

Actual pointer to the media file. The REF element identifies the line as a pointer to media content, while the HREF attribute is the URL to the media file. In this case, the URL uses the MMS protocol, so it points to a Windows Media server.

Media files on your media server are not usually kept in the same location as your HTML documents.

Note the use of the XML-like closing of the element , "]>", instead of "</REF>". Because this element does not have child elements, it closes itself.

</ENTRY>

</ASX>

See Also

- Creating Metafile Playlists
- Example Playlists
- Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

© 2000-2003 Microsoft Corporation. All rights reserved.

An Advanced Playlist

The following playlist example shows how to use a more complete set of playlist elements. When writing your own code, you will need to change all URLs and file names to valid file names that are accessible to your Windows Media Player.

Example Code

<ASX version = "3.0">
  <TITLE>Advanced Playlist Demo</TITLE>
  <ABSTRACT>More Information at this Web site</ABSTRACT>
  <MOREINFO HREF="http://www.microsoft.com/windows/windowsmedia" />
  <BANNER HREF = ".\samples\home.gif">
    <ABSTRACT>MSN Web site</ABSTRACT>
    <MOREINFO HREF = "http://www.msn.com" />
  </BANNER>
  <PARAM name="track" value="1"/>
The following table describes the preceding advanced playlist.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ASX version = &quot;3.0&quot;&gt;</td>
<td>The <strong>ASX</strong> element identifies for the client (Windows Media Player) that this is a Windows Media metafile playlist. The <strong>version</strong> attribute specifies the version number of the metafile elements.</td>
</tr>
<tr>
<td>&lt;TITLE&gt;Advanced Playlist Demo&lt;/TITLE&gt;</td>
<td>The <strong>TITLE</strong> element identifies the title of the playlist as a whole. Windows Media Player displays this metadata as the show title.</td>
</tr>
<tr>
<td>&lt;ABSTRACT&gt;More Information at this Web Site&lt;/ABSTRACT&gt;</td>
<td>The <strong>ABSTRACT</strong> element provides the ToolTip for the show title.</td>
</tr>
<tr>
<td>&lt;MOREINFO HREF = &quot;http://www.microsoft.com/windows/windowsmedia&quot; /&gt;</td>
<td>The <strong>MOREINFO</strong> element links the show title to an URL. Pausing the mouse pointer over the show title accesses the ToolTip, if defined. Selecting the show title will then open the designated URL.</td>
</tr>
<tr>
<td>&lt;BANNER HREF = &quot;..\samples\home.gif&quot;&gt;</td>
<td>The <strong>BANNER</strong> element creates an advertising banner in Windows Media Player. The <strong>HREF</strong> attribute specifies the banner graphic (which must be 194 pixels wide by 32 pixels tall).</td>
</tr>
<tr>
<td>&lt;ABSTRACT&gt;MSN Web site&lt;/ABSTRACT&gt;</td>
<td>The <strong>ABSTRACT</strong> element provides the ToolTip for the <strong>BANNER</strong>.</td>
</tr>
</tbody>
</table>
The MOREINFO element links the BANNER graphic to a URL. Holding the mouse pointer over the BANNER accesses a ToolTip, if defined. Selecting the BANNER will open the designated URL.

The PARAM element defines a custom parameter. The name attribute defines the name of the custom parameter as "track". The value attribute defines the value of "track" to be "1".

Closes the BANNER element

Begins the ENTRY element block. This element defines a clip in a playlist by specifying a link in the REF element. Setting ClientSkip to "no" means that the user cannot fast forward or jump to the next clip.

Creates an advertising banner. The HREF is the banner graphic (194x32 pixels).

Provides the ToolTip for the banner.

Provides a link to the URL for the clip Title text.

A comment. Comments are only be visible when the code is viewed.

Provides the ToolTip for the clip Title text.

Identifies the title of the clip. It is the clip TITLE because it is nested within an ENTRY element. Windows Media Player displays this metadata as the clip title.

Identifies the author of the media clip. This metadata is displayed as the clip AUTHOR by Windows Media Player.

The COPYRIGHT element specifies the copyrights associated with the media clip. Windows Media Player displays this metadata as the clip COPYRIGHT.

A comment, in the same format as XML comments.

URL for the media content. The REF element identifies the line as a pointer to media content. The HREF attribute is the URL to the file.

Note the use of the XML-like closing of the
See Also

- Creating Metafile Playlists
- Example Playlists
- Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

An Example Radio Station Playlist

The following example code illustrates how to create a playlist to scan three rock radio stations. The fictitious
company Adventure Works Radio brand is on the playlist and on all of the individual streams within the playlist. When you write your code, change all URLs and file names to valid file names that are accessible to your Windows Media Player.

A playlist is created for each of the stations. A fourth playlist scans through the first three. The playlists are created to reference dynamically generated advertisements and have Adventure Works Radio branding.

One of the playlists representing a radio station might look like this.

```xml
<ASX version = "3.0">
  <TITLE>Adventure Works Radio</TITLE>
  <MOREINFO href = "http://www.adventure-works.com" />
  <ENTRY clientSkip = "no" skipIfRef = "yes">
    <REF href = "http://www.adventure-works.com/ad.asp" />
  </ENTRY>
  <ENTRY>
    <TITLE>MyWRCK Radio</TITLE>
    <ABSTRACT>MyTown's Best Rock 'n Roll</ABSTRACT>
    <COPYRIGHT>2000 RadioNetwork</COPYRIGHT>
    <MOREINFO href = "http://www.adventure-works.com" />
    <REF href = "http://www.adventure-works.com" />
    <REF href = "http://backup.adventure-works.com" />
  </ENTRY>
</ASX>
```

The playlist can then be constructed of references to the individual playlists.

**Example Code**

```xml
<ASX Version = "3.0">
  <TITLE>Adventure Works Radio Top 3 Rock Stations</TITLE>
  <MOREINFO href = "http://www.adventure-works.com/MyTop3Rocks"/>
  <REPEAT>
    <ENTRY ClientSkip = "no">
      <REF HREF = "http://www.adventure-works.com/ad.asp" />
    </ENTRY>
    <DURATION VALUE="00:00:30" />
    <ENTRYREF HREF = "http://www.adventure-works.com/asx/RadioNetwork.wax"/>
    <DURATION VALUE="00:00:30" />
    <ENTRYREF HREF = "http://www.adventure-works.com/asx/RadioNetwork2.wax"/>
    <DURATION VALUE="00:00:30" />
  </REPEAT>
</ASX>
```

This example would play an ad followed by 30 seconds of each of the three stations referenced, one after the other. This cycle will repeat indefinitely because the `COUNT` attribute of the `REPEAT` element is not defined.

*The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.*

**See Also**

- Creating Metafile Playlists
- Example Playlists
- Metafile Playlists
- Windows Media Metafile Elements Reference
Metafile Extension Guidelines

A file name extension provides an independent software vendor (ISV) with information about the rendering requirements of an application that uses the extension, and enables content authors to target general types of players.

Windows Media metafile extensions are used to identify the format of the Windows Media files that a metafile references. Windows Media metafiles with .wax, .wvx, or .asx extensions reference files with .wma, .wmv, and .asf extensions, respectively. All metafiles, regardless of the file name extension used, have the ASX element tag at the beginning of the file with the version attribute specified.

The following table shows the media file types referenced by each type of metafile file name extension. The columns list media file extensions, the rows list metafile extensions. An X in a column indicates a media file type that can be referenced by a particular metafile file name extension.

<table>
<thead>
<tr>
<th>Extension</th>
<th>.asf</th>
<th>.wma</th>
<th>.wmv</th>
</tr>
</thead>
<tbody>
<tr>
<td>.wvx</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>.wax</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>.asx</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also

- File Name Extensions
- Metafile Playlists
- Windows Media Metafile Guide
Order of Precedence

Not all metafile element attributes are created equal. Some metafile element attributes override other element attributes. Element attributes can be overridden by similar element attributes depending on position and order. Any attributes of a metafile playlist override those contained in a referenced Windows Media file. An attribute that overrides another has higher precedence.

The hierarchy, highest precedence to lowest, is shown in the following table. The highest precedence item is never overridden.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Signed DRM content&quot;</td>
<td>Never overridden.</td>
</tr>
<tr>
<td>REF element scope</td>
<td>Only overridden by signed DRM content.</td>
</tr>
<tr>
<td>ENTRY element scope</td>
<td>Overrides elements of the categories below.</td>
</tr>
<tr>
<td>ASX scope</td>
<td>Overrides media file elements.</td>
</tr>
<tr>
<td>Windows Media file scope</td>
<td>Overridden by all of the above.</td>
</tr>
</tbody>
</table>

- "Signed DRM content" - Digital signature object.
  Attributes of Signed DRM content will override all others. For example, the Copyright information of "Signed DRM content" will not be overridden. It will always be streamed and presented.

- REF element scope
  Attributes of the REF element will override other element attributes, but not signed DRM content.

- ENTRY scope
  Attributes of the ENTRY element will be overridden by the REF element attribute but will override other element attributes. TITLE metadata from the ENTRY element is displayed instead of the title information from the media file.

- ASX scope
  Any properties that are entered in the metafile override those contained in the Windows Media file. Attributes of the ENTRY element override ASX element attributes. While the ENTRY element's referenced media clip is playing, TITLE metadata from the ENTRY element is displayed instead of title information from the ASX element.

- Windows Media file scope
Attributes of the Windows Media file are overridden by any metafile attributes. Media file metadata is displayed only if there is no metadata defined for that element in the metafile.

See Also

- Creating Metafile Playlists
- Metafile Playlists
- Windows Media Metafile Elements Reference
- Windows Media Metafile Guide

Windows Media Metafile Reference

This reference documents elements and file extensions for Windows Media metafiles. The reference is divided into the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Media Metafile Elements Reference</td>
<td>Documents metafile elements, including definitions, attributes and their values, and special conditions related to each element.</td>
</tr>
<tr>
<td>File Name Extensions</td>
<td>Documents metafile file name extensions with rules and guidelines on their use.</td>
</tr>
</tbody>
</table>

Windows Media metafiles are text files that provide information about a file stream and its presentation. The metafiles are based on the Extensible Markup Language (XML) syntax, and are made up of various XML-like elements with their tags and attributes. Each element defines a setting or action for streaming media.

There are two sets of element tags available for metafiles. Client-side metafiles have one set of elements, and server-side metafiles have another set of elements.

If an element tag does not have any child elements (those that modify or are contained within another element), a single slash character (/) can be used at the end of the opening tag in place of a closing tag. If child elements do not appear between the opening and closing tag for an element, they are not child elements for that element, and are ignored or cause an error in the syntax of the metafile.

See Also

- About Windows Media Metafiles
- Windows Media Metafile Guide
Windows Media Metafile Elements Reference

This section contains reference documentation for all Windows Media metafile elements for client-side applications. Reference entries include definitions of the elements, their attributes and values, and special conditions related to each element.

The following metafile elements are supported for client-side applications.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABSTRACT</strong></td>
<td>Contains text that describes the associated ASX, BANNER, or ENTRY element.</td>
</tr>
<tr>
<td><strong>ASX</strong></td>
<td>Defines a file as a Windows Media metafile.</td>
</tr>
<tr>
<td><strong>AUTHOR</strong></td>
<td>Contains the name of the author of a media clip or a Windows Media metafile.</td>
</tr>
<tr>
<td><strong>BANNER</strong></td>
<td>Specifies a URL for a graphic that appears in the display panel of Windows Media Player.</td>
</tr>
<tr>
<td><strong>BASE</strong></td>
<td>Specifies a string that is appended to the front of URLs sent to Windows Media Player.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>Specifies the XML syntax for comments ( &lt;!--...--&gt; ).</td>
</tr>
<tr>
<td><strong>COPYRIGHT</strong></td>
<td>Contains the copyright information for an ASX or ENTRY element.</td>
</tr>
<tr>
<td><strong>DURATION</strong></td>
<td>Specifies the length of time Windows Media Player renders a stream.</td>
</tr>
<tr>
<td><strong>ENDMARKER</strong></td>
<td>Specifies a marker at which Windows Media Player stops rendering the stream.</td>
</tr>
<tr>
<td><strong>ENTRY</strong></td>
<td>Specifies the path for a media clip.</td>
</tr>
<tr>
<td><strong>ENTRYREF</strong></td>
<td>Links to the ENTRY elements in an external Windows Media metafile that has an .asx, .wax, .wvx, or .wmx extension.</td>
</tr>
<tr>
<td><strong>EVENT</strong></td>
<td>Defines a behavior or action taken by Windows Media Player when it receives a script command labeled as an event.</td>
</tr>
<tr>
<td><strong>LOGURL</strong></td>
<td>Instructs the player to submit any log data to the specified URL.</td>
</tr>
</tbody>
</table>
ABSTRACT Element

The ABSTRACT element contains text that describes the associated ASX, BANNER, or ENTRY element.

Syntax

```xml
<ABSTRACT>
  text string
</ABSTRACT>
```

Attributes

None.

Parent/Child Elements
Remarks

If this element appears within an **ASX** element, the text displays as a ToolTip when the mouse hovers over the show title.

If this element appears within an **ENTRY** element, the text displays as a ToolTip when the mouse hovers over the clip title.

If this element appears within a **BANNER** element, the text is displayed as a ToolTip for the banner graphic.

Use only one **ABSTRACT** element per scope. Only the first **ABSTRACT** element within the scope of another element is used when a metafile file is processed. Any subsequent **ABSTRACT** elements in that scope are ignored.

Example Code

```xml
<ASX VERSION="3.0">
  <TITLE>The Title of the Show</TITLE>
  <ABSTRACT>
    Brief description of the show.
  </ABSTRACT>

  <ENTRY>
    <REF HREF="YourMediaFilename.asf" />
    <TITLE>The Title of the Track</TITLE>
    <ABSTRACT>
      Brief description of the track.
    </ABSTRACT>
  </ENTRY>
</ASX>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- [Windows Media Metafile Elements Reference](#)
- [Windows Media Metafile Reference](#)
ASX Element

The **ASX** element defines a file as a metafile.

Syntax

```
<ASX
    VERSION = "number"
    PREVIEWMODE = "YES" | "NO"
    BANNERBAR = "AUTO" | "FIXED"
>
</ASX>
```

**Attributes**

**VERSION** (required)

Decimal number representing the version number of the syntax for the metafile. Set to 3 or 3.0.

**PREVIEWMODE**

Value indicating whether Windows Media Player enters preview mode before playing the first clip.

Must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Windows Media Player enters preview mode before playing the first clip.</td>
</tr>
<tr>
<td>NO</td>
<td>The default value. Windows Media Player does not enter preview mode before playing the first clip.</td>
</tr>
</tbody>
</table>

**BANNERBAR**

Value indicating whether Windows Media Player reserves space for a banner graphic.

Must be one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO</td>
<td>The default value. Windows Media Player reserves space for the banner bar only when a piece of content includes one.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Windows Media Player reserves a fixed space for a banner graphic for every piece of content played, whether or not there is an associated</td>
</tr>
</tbody>
</table>
banner.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>None. The ASX element must be the first element in every metafile.</td>
</tr>
<tr>
<td>Child elements</td>
<td>ABSTRACT, AUTHOR, BANNER, BASE, COPYRIGHT, ENTRY, ENTRYREF, EVENT, MOREINFO, PREVIEWDURATION, PARAM, REPEAT, TITLE</td>
</tr>
</tbody>
</table>

Remarks

The first four characters of a metafile playlist must be <ASX. Other elements defined within the scope of the ASX element, such as TITLE and AUTHOR, are associated with the show information displayed by Windows Media Player.

For Windows Media Player, the syntax version number is 3.0. Windows Media Player supports all previous versions of metafile syntax. Acceptable values for the VERSION attribute include both 3.0 and 3 (with no decimal point).

If the value of the PREVIEWMODE attribute is YES, Windows Media Player immediately enters preview mode before playing the first clip. When Windows Media Player enters preview mode, it previews each clip referenced in the metafile. The PREVIEWDURATION element determines the duration of each preview.

The BANNERBAR attribute defines whether Windows Media Player reserves space for a banner graphic. A banner is a graphic that is displayed in the video display area while media content is playing. (Use the BANNER element to add a banner to the content.) If the value of BANNERBAR is FIXED, Windows Media Player reserves banner space for every piece of media content, whether or not the media content has a banner. If a piece of media content does not have a banner associated with it, the space reserved for one is black. If the value of the BANNERBAR attribute is AUTO, Windows Media Player reserves space for the banner only when the media content includes one.

If you create a metafile with multiple clips (ENTRY or ENTRYREF elements) and set the value of the BANNERBAR attribute to AUTO, Windows Media Player might resize to allow space for a banner graphic for one clip, and then resize again if the next clip does not contain a banner graphic. If you want the size of the window to stay the same (except when the video size changes), use the FIXED value for the BANNERBAR attribute.

The space reserved for a banner graphic is 32 pixels high by 194 pixels wide. The reserved space appears below any rendered video content and 6 pixels above the lower edge of the video area, allowing space for the 6-pixel video area border. The reserved banner space is centered horizontally.

Windows Media Player renders the graphic beginning in the leftmost pixel of the banner space. If the graphic fills the entire space, it will appear centered horizontally. Otherwise there will be trailing space. Note that the minimum width of Windows Media Player is always wider than the size of the video clip, regardless of the value of the BANNERBAR attribute.

Example Code

```xml
<ASX VERSION="3.0" PREVIEWMODE="YES" BANNERBAR="auto" />
```
AUTHOR Element

The AUTHOR element contains the name of the author of a Windows Media metafile or media clip.

Syntax

```
<AUTHOR>
  text string
</AUTHOR>
```

Attributes

None.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, ENTRY</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

This element contains a text string representing the name of the author of a Windows Media metafile or media clip. You can use the AUTHOR element within the ASX element and within ENTRY elements.

If this element appears in the ASX element, the text is displayed as Show information.
If this element appears in an **ENTRY** element, the text is displayed as the clip author.

Each parent **ASX** and **ENTRY** element should contain at most one child **AUTHOR** element. Multiple **AUTHOR** elements after the first will be ignored and will not be displayed.

**Example Code**

```xml
<ASX VERSION="3.0">
  <AUTHOR>Neal S.</AUTHOR>   <!-- Show author -->
  <ENTRY>
    <REF HREF="mms://example.microsoft.com/clip1.asf" />
    <AUTHOR>Robert P.</AUTHOR>  <!-- Clip author -->
  </ENTRY>
</ASX>
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

---

**BANNER Element**

The **BANNER** Element defines a URL to a graphic file that will appear in the display panel.

**Syntax**

```xml
<BANNER
  HREF = "URL"
>
</BANNER>
```

**Attributes**

- **HREF** (required)

  URL to a graphic file that is displayed in the display panel.
Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, ENTRY</td>
</tr>
<tr>
<td>Child elements</td>
<td>ABSTRACT, MOREINFO</td>
</tr>
</tbody>
</table>

Remarks

This element defines a URL to a graphic file that appears in the Windows Media Player display panel, beneath the video content. If the media is audio only, the banner graphic is displayed by itself. Windows Media Player reserves a space 32 pixels high by 194 pixels wide (the banner bar) for the graphic. If the graphic defined in the URL is smaller than that, it displays at its original size. If the graphic is larger than the reserved space, Windows Media Player will crop the image to fit the space.

You can use an ABSTRACT element within the scope of the BANNER element to display text as a ToolTip when the user pauses the mouse pointer over the banner graphic. A MOREINFO element within a BANNER element defines a URL to which the user is taken when the user clicks the banner graphic. (The URL can be any path or protocol, such as an e-mail link, a Hypertext Transfer Protocol (HTTP) URL to a Web site, or even a Microsoft JScript command.) When the pointer is moved over the graphic, the graphic becomes embossed and looks like a button.

A BANNER element defined for an ASX element displays while all clips in the playlist are playing. A BANNER element defined in an ENTRY element displays only while that clip is playing, and during that time overrides any banner defined within the parent ASX element. You can specify how Windows Media Player reserves space for the banner by setting the BANNERBAR attribute of the ASX element.

Banner images are not supported with DRM files or when the player is embedded in a Web page.

Example Code

The following is an example of a BANNER element without child elements:

```xml
<BANNER HREF="http://sample.microsoft.com/art/banner1.bmp" />
```

The following is an example of a BANNER element containing ABSTRACT and MOREINFO elements.

```xml
<BANNER HREF="http://www.proseware.com/logos/banner1.bmp">
  <ABSTRACT>Click here to go to our Web site.</ABSTRACT>
  <MOREINFO HREF="http://sample.microsoft.com" />
  <!-- The text in the Abstract element displays as a ToolTip when the mouse hovers over the banner graphic. When a user clicks the banner, the URL given in the MoreInfo element opens in the browser. -->
</BANNER>
```

Requirements

Windows Media Player version 7.0 or later.

See Also
BASE Element

The **BASE** element defines a URL string appended to the front of URLs sent to Windows Media Player.

Syntax

```xml
<BASE
   HREF = "URL"
/>
```

Attributes

- **HREF** (required)

  The string appended to the front to URLs sent to Windows Media Player. The default value is the null string ("").

Parent/Child Elements

**Hierarchy**

- Parent elements: **ASX, ENTRY**
- Child elements: None

Remarks

This element defines a URL string appended to the front of URL-flip URLs sent to Windows Media Player. URL-flipping is a scripting mechanism that causes Windows Media Player to open a new URL in a browser or browser frame when it receives a script command.

The **BASE** element is similar to a home directory for relative links. It only affects URLs sent using script commands as part of the content stream that is playing in Windows Media Player.

A **BASE** element defined as the child of an **ASX** element becomes the default for the entire metafile. A **BASE** element defined in an **ENTRY** element overrides any **BASE** element in the parent **ASX** element (for that **ENTRY** element only).
Note  If the HREF attribute does not end with a / character, Windows Media Player appends one to the end of the string.

Example Code

```xml
<BASE HREF="http://msdn.microsoft.com/" />
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

Comments

Add comments to metafiles by following Extensible Markup Language (XML) syntax. Comments begin with "<!--" and end with "-->".

Syntax

```xml
<!--Enter your comment text here.-->
```

Remarks

Comments can appear anywhere except within element content (between element open and close tags, < >). They are not part of the document's character data and are ignored when the metafile is parsed.

Example Code

```xml
<ASX version = "3.0">
<!-- This information is only visible when editing a metafile file. -->
<!--<BANNER HREF="c:\wmsdk\wmssdk\samples\dhtml\asfbutton3.gif">
</BANNER>-->
<ENTRY>
  <REF HREF = "mms://proseware.com/pubpt/filename.asf" />
</ENTRY>
<ENTRY>
```

© 2000-2003 Microsoft Corporation. All rights reserved.
Adding Copyright Characters to Metafiles

The characters for copyright and trademark registration symbols (© or ®) may not display properly if the metafile is not encoded using the UTF-8 encoding scheme. In this case, to display either of these symbols properly for all users, you can use the ASCII equivalents (c) and (r) inside the COPYRIGHT element, as shown in the following code.

```xml
<COPYRIGHT>Copyright (c)1998, Microsoft Corporation</COPYRIGHT>
```

If the metafile is encoded using UTF-8, copyright and trademark symbols will display correctly.

See Also

- Windows Media Metafile Elements Reference

COPYRIGHT Element

The COPYRIGHT element defines a text string specifying the copyright information for an ASX or ENTRY.
element.

Syntax

```xml
<COPYRIGHT>
text string
</COPYRIGHT>
```

Attributes

None.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, ENTRY</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

This element defines a text string that specifies the copyright information for an ASX or ENTRY element.

When this element appears within an ASX element, the copyright string is displayed only as Show information. When this element appears within an ENTRY element, the text is displayed as clip information. Each parent ASX and ENTRY element should contain at most one child COPYRIGHT element. Multiple COPYRIGHT elements after the first will be ignored and will not be displayed.

The characters for copyright and trademark registration symbols (© or ®) may not display properly if the metafile is not encoded using the UTF-8 encoding scheme. In this case, to display either of these symbols properly for all users, you can use the ASCII equivalents (c) and (R) instead.

If the metafile is encoded using UTF-8, copyright and trademark symbols will display correctly.

Example Code

```xml
<COPYRIGHT>Copyright (c)1998, Microsoft Corporation</COPYRIGHT>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

© 2000-2003 Microsoft Corporation. All rights reserved.
DURATION Element

The DURATION element defines the length of time Windows Media Player will render the associated playlist entry.

Syntax

```
<DURATION VALUE="hh:mm:ss.fract" />
```

Attributes

**VALUE** (required)

The length of time, in hours, minutes, seconds, and hundredths of a second, that an entry is rendered by Windows Media Player. The default value is the entire length of the entry. If the entry is an image file, a duration value must be specified.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ENTRY, REF</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

This element defines the length of time a stream is to be rendered. If the **VALUE** attribute exceeds the length of the content stream, the stream terminates at its normal end-point.

This element can appear either within a **REF** element or within an **ENTRY** element. However, a **DURATION** element defined within a **REF** element overrides one that appears within the **REF** element's parent **ENTRY** element.

The **DURATION** element overrides a **PREVIEWDURATION** element.

Example Code

```
<DURATION VALUE="00:00:30" /> <!-- 30 seconds -->
<DURATION VALUE="1:01.5" /> <!-- 61.5 seconds -->
```

Requirements
The **ENDMARKER** element specifies a marker at which Windows Media Player will stop rendering the stream.

**Syntax**

```xml
<ENDMARKER
    NUMBER = "marker number" |
    NAME = "marker name"
/>```

**Attributes**

**NUMBER**

The number of a numeric marker in the index. The default value is the end of the content.

**NAME**

The name of a named marker in the index. The default value is the end of the content.

**Parent/Child Elements**

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ENTRY, REF</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

**Remarks**

This element specifies the marker where Windows Media Player is to stop rendering the stream defined in the
parent ENTRY or REF element.

**Note** Use the ENDMARKER element with either the NUMBER or NAME attribute, but not both.

In preview mode, reaching an end marker stops the preview, even if the time specified by the PREVIEWDURATION element has not elapsed. The ENDMARKER element also takes precedence over the DURATION element.

An ENDMARKER element defined within a REF element takes precedence over an ENDMARKER defined within the REF element's parent ENTRY element.

If the marker specified by an ENDMARKER element occurs earlier in the stream than the marker defined by a STARTMARKER element, no content plays, but no error is generated.

**Example Code**

```xml
<ENDMARKER NUMBER="17" />
<ENDMARKER NAME="Marker_StopHere" />
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**ENTRY Element**

The ENTRY element specifies a Windows Media file to render as a clip.

**Syntax**

```xml
<ENTRY
    CLIENTSKIP = "YES" | "NO"
    SKIPIFREF = "YES" | "NO"
>
</ENTRY>
```
Attributes

CLIENTSKIP

Value indicating whether the user can skip forward past the clip.

Possible values include the following.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Default. User can skip forward past the clip.</td>
</tr>
<tr>
<td>NO</td>
<td>User cannot skip forward past the clip.</td>
</tr>
</tbody>
</table>

SKIPIFREF

Value indicating whether Windows Media Player should skip this clip when the ENTRY element is included in a second metafile through the use of an ENTRYREF element.

Possible values include the following.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Windows Media Player will ignore this entry, if referenced through an ENTRYREF element.</td>
</tr>
<tr>
<td>NO</td>
<td>Default. Windows Media Player will not ignore this entry.</td>
</tr>
</tbody>
</table>

Parent/Child Elements

Hierarchy | Elements
---|---
Parent elements | ASX, EVENT, REPEAT
Child elements | ABSTRACT, AUTHOR, BANNER, BASE, COPYRIGHT, DURATION, ENDMARKER, MOREINFO, PARAM, PREVIEWDURATION, REF, STARTMARKER, STARTTIME, TITLE

Remarks

This element is the fundamental construct in a Windows Media metafile. The ENTRY element and its associated attributes define meta-information for a single, logical piece of content, called a clip. Child elements within the scope of an ENTRY element define media content for Windows Media Player to open (REF), information about the clip that Windows Media Player will display as text (AUTHOR, COPYRIGHT, TITLE), and other settings related to the clip. The REF child element links the content to be streamed for the parent ENTRY element. Though the script will not break, the ENTRY is meaningless without a REF child.

If the value of the CLIENTSKIP attribute is NO, the user cannot skip forward past the piece of content defined by the ENTRY element. This does not work if the child REF element references another metafile. Nested metafiles should be referenced using the ENTRYREF element.
The **SKIPIFREF** attribute pertains only to **ENTRY** elements that are included in a second metafile through the use of an **ENTRYREF** element. The **ENTRYREF** element references another metafile for logical inclusion in the current file. If the value of the **SKIPIFREF** attribute for an **ENTRY** element from the referenced metafile file is YES, Windows Media Player ignores this pulled-in entry, and moves on to the next **ENTRY** element, if any. The next **ENTRY** element can be the next entry in the original file, or the next entry in the metafile referenced in the **ENTRYREF** element.

**Example Code**

```xml
<ASX VERSION="3.0">
  <TITLE>Example Windows Media Player Show</TITLE>
  
  <ENTRY>
    <TITLE>Example Clip</TITLE>
    <REF HREF="http://example.microsoft.com/media.asf" />
  </ENTRY>

  <ENTRY>
    <TITLE>Another Clip</TITLE>
    <REF HREF="http://example.microsoft.com/more_media.asf" />
  </ENTRY>

</ASX>
```

**Requirements**

Windows Media Player version 7.0 or later.

**See Also**

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

---

© 2000-2003 Microsoft Corporation. All rights reserved.

---

**ENTRYREF Element**

The **ENTRYREF** element points to the **ENTRY** elements in an external metafile.

**Syntax**

```xml
<ENTRYREF
  HREF = "URL"
/>```
Attributes

**HREF** (required)

URL to a referenced metafile.

**CLIENTBIND**

This attribute is no longer supported.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, EVENT, REPEAT</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

This element points to the contents of an external metafile. Windows Media Player processes the **ENTRY** elements of the referenced metafile as if they were included in the primary metafile at the position of the **ENTRYREF** element. However, it skips any **ENTRY** elements in the referenced metafile that have the **SKIPIFREF** attribute set to YES.

Windows Media Player 7.0, 7.1, and Windows Media Player for Windows XP ignore any **ENTRYREF** elements in the referenced metafile. Thus, metafiles can only be nested one level deep using these Player versions. Windows Media Player also ignores the **ASX** element in the referenced file along with its attributes. Windows Media Player 9 Series or later supports nesting metafiles up to five levels deep.

The URL specified in the **HREF** attribute becomes the base URL for any relative URLs in the external metafile. This URL is used when an entry from the external metafile is playing and the user adds it to the **Media Library**.

Example Code

```
<ASX VERSION="3.0">
  <TITLE>Example of Using EntryRef</TITLE>
  <ENTRYREF HREF="http://sample.microsoft.com/metafile.asx" />
</ASX>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- [Windows Media Metafile Elements Reference](#)
- [Windows Media Metafile Reference](#)

Previous Next
EVENT Element

The EVENT element defines a behavior or action taken by Windows Media Player when it receives a script command labeled as an event.

Syntax

```
<Event
    NAME = "text string"
    WHENDONE = "RESUME" | "NEXT" | "BREAK"
>
</Event>
```

Attributes

**NAME** (required)

The name of the event.

**WHENDONE** (required)

A value that defines what Windows Media Player does after playing the referenced content.

The following values are possible.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESUME</td>
<td>The current entry (the clip interrupted by the event) resumes playing. If the content is stored content, it resumes at the same point where it stopped; if the content is broadcast, it resumes at the current position.</td>
</tr>
<tr>
<td>NEXT</td>
<td>The next ENTRY element plays as if the event had not occurred and Windows Media Player had reached the end of the current clip.</td>
</tr>
<tr>
<td>BREAK</td>
<td>If the current entry is within a REPEAT loop, the loop terminates as if the repeat count had been completed. Otherwise, Windows Media Player jumps to the end of the playlist as if the final entry had completed as usual.</td>
</tr>
</tbody>
</table>

Parent/Child Elements

**Hierarchy Elements**
Remarks

This element defines a behavior or action taken by Windows Media Player when it receives a script command labeled as an event. An event is a particular type of script command embedded in a stream sent to Windows Media Player that consists of a double string. The first string is the word "event", and the second string is the event name. The event name in the second string must match the event name defined in the metafile. (The match is not case-sensitive.) Events can be sent to Windows Media Player receiving a real-time stream, or can be saved in a file with an .asf, .wma, or .wmv file name extension, that gets delivered as an on-demand unicast stream. When Windows Media Player receives the script command, it processes the event as defined by the EVENT element.

This element defines a scope of ENTRY or ENTRYREF elements that are processed whenever Windows Media Player receives the script command with the named event. The ENTRYREF can be a URL that points to an ASP page. With this element, you can specify a behavior for stream switching in near real-time, as opposed to pre-authored stream changes using references to other pieces of content or Windows Media metafiles.

When you use ASP pages to generate playlists, you must specify a value for the Response.ContentType property and the Response.expires property in the ASP page because of latency issues with Windows Media Player. The Response.ContentType must be a valid file name extension for Windows Media metafiles. Valid types include asf, asx, wma, wax, wmv, and wvx.

See the Platform SDK for details about using the Response object in Active Server Pages.

This element can appear anywhere within the ASX element. If multiple EVENT elements within an ASX element have identical values for their NAME attributes, Windows Media Player uses the first occurrence within the ASX element, and ignores all others. When EVENT elements have distinct NAME attributes, their order within the ASX element does not matter.

Windows Media Player discards events it receives while processing another event. Nesting of events is not supported. When Windows Media Player is in preview mode, event content is not constrained by the PREVIEWDURATION element; the full length of event content can play even if the preview duration for the active ENTRY element expires prior to the end of the event.

Example Code

In this example, when Windows Media Player receives the script command EVENT and command string "Adlink" in the streaming media it is rendering, it searches the playlist for an EVENT NAME "Adlink". Windows Media Player switches from the stream it is rendering and plays the content referenced in the EVENT, "http://example.microsoft.com/adlink.htm".

ENTRY attribute CLIENTSKIP is set to NO to prevent the EVENT clip from being skipped. It must be played.

<ASX VERSION="3.0">
  <ENTRY CLIENTSKIP="NO">
    <REF HREF="http://example.microsoft.com/clip1.asf" />
  </ENTRY>
  <EVENT NAME="Adlink" WHENDONE="RESUME">
    <ENTRYREF HREF="http://example.microsoft.com/adlink.htm" />
  </EVENT>
</ASX>
The script `WHENDONE="RESUME"` instructs Windows Media Player to resume playing the previous media it switched from as soon as Adlink.asf is finished.

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference
- Windows Media Player Object Model

LOGURL Element

The `LOGURL` element instructs the player to submit any log data to the specified URL.

Syntax

```xml
<LOGURL
   HREF = "URL"
/>
```

Attributes

- `HREF` (required)

URL to a host that is able to process logging requests.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, ENTRY</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>
Remarks

The LOGURL element enables a client metafile playlist to send additional logging information to specified servers. Logging information is automatically sent to the origin server of a playlist when it is opened and to each LOGURL specified for the ASX element, if any are present. Logging information is also sent to each LOGURL specified for an ENTRY element when that entry is reached. The URL specified in the HREF attribute of a LOGURL element must be the address of a host that is able to process logging requests. The URL can be any valid HTTP URL. The URL can also indicate the location of a CGI script.

The only valid protocols for a LOGURL element are HTTP and HTTPS.

A LOGURL element within the scope of an ASX element is applicable only to the metafile in which it resides, regardless of whether that metafile is referenced from another metafile. A LOGURL element forces the submission of log data for all content streamed from within its defined scope and only for content streamed from within its defined scope. Log data will be submitted to the origin server and to all URLs listed in every LOGURL element in scope. Log data will be submitted only once to each listed URL, even if the same URL is listed more than once in a given scope. A repeat of an ENTRY would result in another submission to the listed URLs.

There is no limit to the number of LOGURL elements in a metafile playlist.

Example Code

<ASX VERSION="3.0">
  <TITLE>Example Media Player Show</TITLE>
  <LOGURL HREF="http://example.microsoft.com/info/showlog.asp?whatsup" />
  <ENTRY>
    <REF href="mms://ucast.proseware.com/Media1.asf" />
    <LOGURL HREF="http://www.proseware.com/cgi-bin/logging.pl?SomeArg=SomeVal"/>
  </ENTRY>
</ASX>

The following are examples of valid URLs.

URL of an ISAPI application:

http://www.proseware.com/logs/WMSLogging.dll

URL of a CGI script:

http://www.proseware.com/cgi-bin/My_WMS_Logging_Script.pl

A valid HTTP URL:


Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
MOREINFO Element

The MOREINFO element specifies a URL to a Web site, e-mail address, or script command associated with a show, clip, or banner.

Syntax

```xml
<MOREINFO
  HREF = "URL"
  TARGET = "Frame"
/>
```

Attributes

- **HREF** (required)
  
  URL to a Web site, e-mail address, or script command.

- **TARGET**
  
  Value defining the frame or window in which the browser will open the page defined by the HREF attribute.

This can be a string containing a window name, or one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_blank</td>
<td>The document loads in a new browser window.</td>
</tr>
<tr>
<td>_self</td>
<td>The document loads in the same frame as the current document containing the Windows Media Player control.</td>
</tr>
<tr>
<td>_parent</td>
<td>The document loads in the immediate parent frame of the current frame, or the current frame if there is no parent frame.</td>
</tr>
<tr>
<td>_top</td>
<td>The document loads in the full browser window, replacing all other frames or documents.</td>
</tr>
</tbody>
</table>

Parent/Child Elements
**Remarks**

This element specifies a URL to a Web site, e-mail address, or script command. The user can access the target of the URL by clicking on the graphic or text associated with the **MOREINFO** element. The details depend on the parent element of the **MOREINFO** element:

- If the **MOREINFO** element is the child of an **ASX** element, the user can access the URL by clicking the show title.
- If the **MOREINFO** element is the child of an **ENTRY** element, the user can access the URL by clicking the clip title.
- If the **MOREINFO** element is the child of a **BANNER** element, the user can access the URL by clicking the banner graphic.

If the **HREF** attribute specifies a URL to a Web site, the browser opens and navigates to the URL. If the **HREF** attribute specifies an e-mail address, the user's e-mail application starts. If the **HREF** attribute specifies a script command, the script command executes in the browser.

**Note** If the **MOREINFO** element appears within an **ASX** or **ENTRY** element, when the mouse cursor is held over the title of the show (for an **ASX** element) or clip (for an **ENTRY** element), the URL defined in the **HREF** attribute can be selected and accessed by Windows Media Player.

The **TARGET** attribute defines the frame or window in which the browser will open the page defined by the **HREF** attribute. The values for this attribute follow standard HTML syntax and definitions. In the case of a control embedded in a Web page, if no **TARGET** attribute is defined, the URL loads the current browser window and replaces the existing page, which means the content stops playing. Therefore, it is recommended that you always specify a **TARGET** attribute when embedding the Windows Media Player control in a Web page.

If the metafile is opened in the stand-alone Windows Media Player, the **TARGET** attribute is ignored, and the URL loads in an existing browser window. If there is no browser window currently open, the URL loads in a new browser window.

**Example Code**

```xml
<ASX VERSION="3.0">
  <TITLE>Example Media Player Show</TITLE>
  <MOREINFO HREF="http://example.microsoft.com/info/show_info.htm" />

  <ENTRY>
    <TITLE>Example Clip</TITLE>
    <MOREINFO HREF="http://example.microsoft.com/info/clip1_info.htm" />
    <REF HREF="mms://example.microsoft.com/media.asf" />
  </ENTRY>

</ASX>
```

**Requirements**
PARAM Element

The PARAM Element defines a custom parameter associated with a clip or show.

Syntax

```xml
<PARAM
    NAME = "parameter name"
    VALUE = "parameter value"
/>
```

Attributes

NAME

Name used to access the parameter value. The name can be any valid string. The following strings have already been defined.

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowShuffle</td>
<td>The VALUE attribute specifies whether the metafile playlist allows the Windows Media Player shuffle feature to play the entries in random order. The VALUE attribute can be set to &quot;Yes&quot; or &quot;No&quot;. The default value is &quot;No&quot;.</td>
</tr>
<tr>
<td>Encoding</td>
<td>The VALUE attribute is set to &quot;utf-8&quot; to indicate that the metafile is a Unicode (UTF-8) encoded file.</td>
</tr>
<tr>
<td>HTMLView</td>
<td>The VALUE attribute specifies a URL that displays in the Now Playing pane for the duration of the playlist or the current entry depending on whether the parent element is the ASX element or an ENTRY element.</td>
</tr>
<tr>
<td>Log:FieldName[:NameSpace]</td>
<td>The VALUE attribute specifies the value that the indicated log field will be set to. The :NameSpace portion of the NAME attribute is optional.</td>
</tr>
</tbody>
</table>
Prebuffer

The **VALUE** attribute specifies whether the next playlist entry begins buffering before the end of the current entry in order to enable a seamless transition. The **VALUE** attribute can be set to "true" or "false".

ShowWhileBuffering

The **VALUE** attribute specifies whether an image file referenced by the current **ENTRY** element continues to display past its specified duration time while the next playlist entry is buffered. The **VALUE** attribute can be set to "true" or "false".

### VALUE

Value associated with this parameter. It can be either a numeric or string value.

### Parent/Child Elements

#### Hierarchy

Parent elements

ASX, ENTRY

Child elements

None

### Remarks

This element allows users to place additional information about each clip inside the **ENTRY** element that contains it. To retrieve metadata information specified in the playlist **ENTRY**, use the **Media.GetItemInfo** method. The **Media.GetItemInfo** method retrieves the value of the **VALUE** attribute, given the name of the parameter. Previous versions of Windows Media Player retrieve metadata information specified in the playlist **ENTRY**, using the **GetMediaParameter** method given the name of the parameter and an index number for the entry.

A parameter can also be associated with the show rather than an individual clip, by placing this element directly after the **ASX** tag. These items are referenced by their names and an index value beginning with zero.

**Note** This **ASX** element is only available for Windows Media Player version 6.01 and later. The standard installation of Microsoft Internet Explorer 5 includes a compatible version of Windows Media Player.

### Example Code

```xml
<ASX VERSION="3.0">
  <TITLE>Example Media Player Show</TITLE>
  <PARAM NAME="Director" VALUE="Jane D." />

  <ENTRY>
    <TITLE>Example Clip</TITLE>
    <REF HREF="http://sample.microsoft.com/media.asf" />
    <PARAM NAME="Location" VALUE="North America" />
    <PARAM NAME="Release Date" VALUE="March 1998" />
  </ENTRY>

  <ENTRY>
    <TITLE>Another Clip</TITLE>
    <REF HREF="http://sample.microsoft.com/more_media.asf" />
    <PARAM NAME="Location" VALUE="Japan" />
    <PARAM NAME="Release Date" VALUE="December 1996" />
  </ENTRY>
</ASX>
```
Requirements

Windows Media Player version 7.0 or later.

Windows Media Player 9 Series is required for the predefined NAME attributes.

See Also

- Displaying Web Pages in Windows Media Player
- Logging Stream Data
- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

PREVIEWDURATION Element

The PREVIEWDURATION element defines the length of time a clip plays in preview mode.

Syntax

```xml
<PREVIEWDURATION
  VALUE = "hh:mm:ss.fract"
/>```

Attributes

- VALUE (required)

Length of time, in hours, minutes, seconds, and hundredths of seconds, that the clip plays in preview mode.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX, ENTRY, REF</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks
This element defines the length of time that a clip plays in preview mode. If this element appears within an ENTRY element or a REF element, it applies to the clip defined by that element. If it appears within the scope of an ASX element, it applies to every clip in the metafile. A PREVIEWDURATION element in a REF element takes precedence over one in an ENTRY element, and either takes precedence over a PREVIEWDURATION element in an ASX element. If no PREVIEWDURATION element is defined for a clip, the default preview time is 10 seconds.

If there is a STARTTIME or STARTMARKER element for the clip, Windows Media Player renders the clip starting at the point defined by one of these elements; otherwise it renders from the beginning of the clip. The clip stops normally if it is shorter than the time defined by the PREVIEWDURATION element.

The DURATION element overrides a PREVIEWDURATION element.

Example Code

<PREVIEWDURATION VALUE="0:30.0" />

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

© 2000-2003 Microsoft Corporation. All rights reserved.

REF Element

The REF element specifies a URL for a piece of media content.

Syntax

<REF
  HREF = "URL"
>
</REF>

Attributes

HREF (required)
URL to any piece of media content supported by Windows Media Player.

**Parent/Child Elements**

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ENTRY</td>
</tr>
<tr>
<td>Child elements</td>
<td>DURATION, ENDMARKER, PREVIEWDURATION, STARTMARKER, STARTTIME</td>
</tr>
</tbody>
</table>

**Remarks**

This element specifies a URL for a piece of media content. The URL can point to any media type supported, using any protocol supported by Windows Media Player.

The media types supported include still images such as GIF and JPG images and Flash files with a .swf file name extension. These media types are useful for including advertising content within a playlist. With image files and Flash files that play in a loop, you must also specify the amount of time to display the media item by including a **DURATION** element within the **REF** element. If you want an image to continue displaying while the next entry in the playlist is buffered, include a **PARAM** element within the **ENTRY** element, set its **name** attribute to ShowWhileBuffering, and set its **value** attribute to true.

To reference content on a CD-ROM or a DVD-ROM that allows it, the **wmpcd** and **wmpdvd** protocols are provided. For example, setting the **HREF** attribute to "wmpdvd://f/5/3" will play chapter 3 of title 5 on a DVD, but only if the DVD has been authored to allow it.

Applications that open digital media from behind a firewall will have better performance when opening the media items if the address is specified using the domain name server (DNS) name instead of the IP address.

The most common use of this element is for URL rollover. If Windows Media Player is unable to open a piece of media defined in a **REF** element, it tries the URL in the next **REF** element. Once Windows Media Player opens media content from a URL defined within the scope of one **ENTRY** element, it ignores subsequent **REF** tags within that **ENTRY** element. After the piece of content is done playing, Windows Media Player moves on to the next **ENTRY** element, if any.

**Important** Once Windows Media Player establishes a connection to a referenced piece of content, it ignores all other **REF** elements in that **ENTRY**, whether the connection terminates normally or abnormally.

If the media item referenced is an image file, the **DURATION** element must be used to specify the display time for the image.

**Note** Attempting to play Flash media that includes sound with the first frame may yield unexpected results. You should author Flash content to play sound starting no earlier than the second frame.

**Example Code**

```xml
<ASX VERSION="3.0">
  <ENTRY>
    <TITLE>Example Clip</TITLE>
    <REF HREF="mms://example.microsoft.com/selection1.asf" />
    <REF HREF="mms://sample.microsoft.com/mirror/selection1.asf" />
  </ENTRY>
</ASX>
```
REPEAT Element

The REPEAT element defines the number of times Windows Media Player repeats one or more ENTRY or ENTRYREF elements.

Syntax

```
<REPEAT
  COUNT = "integer"
>
</REPEAT>
```

Attributes

COUNT

Integer representing the number of times Windows Media Player repeats the ENTRY and ENTRYREF elements within this element's scope.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
</tr>
<tr>
<td>Child elements</td>
</tr>
</tbody>
</table>

Remarks

This element defines the number of times Windows Media Player repeats, or loops through, the clips defined by
the ENTRY and ENTRYREF elements within this element's scope. Only the first REPEAT element in a metafile is valid; subsequent REPEAT elements are ignored.

If no COUNT attribute is defined, the content in the associated ENTRY and ENTRYREF elements repeats an infinite number of times. A value of zero causes Windows Media Player to ignore the REPEAT element and play the content once.

Example Code

```xml
<ASX VERSION="3.0">
  <ENTRY>
    <REF HREF="mms://example.microsoft.com/clip1.asf" />
    <!-- This clip plays once. -->
  </ENTRY>

  <REPEAT COUNT="2">
    <ENTRY>
      <REF HREF="mms://example.microsoft.com/clip2.asf" />
      <REF HREF="mms://example.microsoft.com/clip3.asf" />
      <!-- These clips play twice. -->
    </ENTRY>
  </REPEAT>
</ASX>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

SKIN Element

The SKIN element specifies a URL to a border.

Syntax

```
<Skin
  HREF = "URL"
/>```
Attributes

**HREF (required)**

URL for a compressed border file with a file extension .wmz. For Windows Media Player 9 Series or later, this value can reference only border files on the user's hard disk located in the same directory as the metafile playlist. See the example code.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ASX</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

The **SKIN** element is used to create a border, which is similar to a skin but is displayed in the *Now Playing* area of the full mode Windows Media Player. The **SKIN** element is used only for borders, which appear inside the full mode Windows Media Player, and not for regular skins, which entirely replace the compact mode Windows Media Player.

In a Windows Media Download Package (with a .wmd file name extension), the **SKIN** element enables a border to have content and link to other sites. The Windows Media Download Package is a compressed file that contains a border file and a Windows Media metafile. The border file (with a .wmz file name extension) is compressed, and includes a skin definition file (with a .wms file name extension).

A **SKIN** element has three components:

- A skin
- Some content
- A metafile

Skins included with Windows Media Download Packages must be rectangular in shape. Creating borders with skins that are not rectangular may yield unexpected results.

Example Code

```xml
<ASX version = "3.0">
  <TITLE>A Skin Element</TITLE>
  <SKIN HREF = "YourTest.wmz" />
  <ENTRY>
    <PARAM name="YourAlbumTitle" value="YourTitle.jpg"/>
    <PARAM name="link" value="http://www.proseware.com"/>
    <TITLE>(The Artist)-YourTitle</TITLE>
    <REF HREF="(The Artist)-YourSongTitle.wma"/>
  </ENTRY>
</ASX>
```

Requirements
STARTMARKER Element

The STARTMARKER element specifies a marker from which Windows Media Player will start rendering the stream.

Syntax

```
<STARTMARKER
    NUMBER = "marker number"
    NAME = "marker name"
/>
```

Attributes

**NUMBER**

The number of a numeric marker in the index. The default value is the beginning of the content if on-demand, or the current position in the stream if real-time.

**NAME**

The name of a named marker in the index. The default value is the beginning of the content if on-demand, or the current position in the stream if real-time.

Parent/Child Elements

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent elements</td>
<td>ENTRY, REF</td>
</tr>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>
Remarks

This element specifies the marker from which Windows Media Player is to start rendering the stream defined in the parent ENTRY or REF element.

**Note** Use this element with either the NUMBER or NAME attribute, but not both.

A STARTMARKER element defined within a REF element takes precedence over a STARTMARKER element defined within the REF element's parent ENTRY element. A STARTMARKER element also takes precedence over a STARTTIME element.

If the marker specified by a STARTMARKER element occurs later in the stream than the marker defined by an ENDMARKER element, no content plays, but no error is generated.

Example Code

```xml
<STARTMARKER NUMBER="14" />
<STARTMARKER NAME="Marker_StartHere" />
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

STARTTIME Element

The STARTTIME element defines a time index from which Windows Media Player will start rendering the stream.

Syntax

```xml
<STARTTIME
   VALUE = "hh:mm:ss.fract"
/>  
```

Attributes
VALUE (required)

The time index, in hours, minutes, seconds, and hundredths of seconds, from which Windows Media Player starts playing a stream defined in the associated element.

Parent/Child Elements

Hierarchy            Elements
Parent elements      ENTRY, REF
Child elements       None

Remarks

This element defines a time index into the content where Windows Media Player is to start rendering the stream. This element can be used only with stored, on-demand content that has been indexed.

Example Code

<STARTTIME VALUE="1:30.0" />

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

© 2000-2003 Microsoft Corporation. All rights reserved.

TITLE Element

The TITLE element defines a text string specifying the title for an ASX or ENTRY element.

Syntax

<TITLE>
   text string
</TITLE>
Attributes

None.

Parent/Child Elements

Hierarchy

<table>
<thead>
<tr>
<th>Parent elements</th>
<th>ASX, ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child elements</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

This element defines a text string that specifies the title of an ASX or ENTRY element. The title is displayed in the display panel and Properties dialog box.

When this element appears within an ASX element, the text is displayed as Show information. When this element appears within an ENTRY element, the text is displayed as Clip information.

If a MOREINFO element is also used with the associated (parent) element, it is the title text that the user clicks to access the URL defined in the MOREINFO element.

Each parent ASX and ENTRY element should contain at most one child TITLE element. Multiple TITLE elements after the first will be ignored and will not be displayed.

Example Code

```xml
<ASX VERSION="3.0">
  <TITLE>Title of the Show</TITLE>
  <ENTRY>
    <TITLE>Title of the Clip</TITLE>
    <REF HREF="mms://example.microsoft.com/clip1.asf" />
  </ENTRY>
</ASX>
```

Requirements

Windows Media Player version 7.0 or later.

See Also

- Windows Media Metafile Elements Reference
- Windows Media Metafile Reference

© 2000-2003 Microsoft Corporation. All rights reserved.
File Name Extensions

There are specific guidelines for the use of file name extensions for Windows Media metafiles. Windows Media metafile extensions are used to identify the different types of Windows Media files. A file name extension provides an Independent Software Vendor (ISV) with information about the rendering requirements of an application that uses a particular extension, and enables content authors to target general types of media players.

The file name extension guidelines are listed in the following table. It is recommended that a file's MIME type, located in the file header, be used for file-type identification.

<table>
<thead>
<tr>
<th>File name extension</th>
<th>MIME type</th>
<th>File content</th>
</tr>
</thead>
<tbody>
<tr>
<td>.wma</td>
<td>audio/x-ms-wma</td>
<td>Windows Media file, with audio content only. Typically used to download and play files or to stream content.</td>
</tr>
<tr>
<td>.wmv</td>
<td>video/x-ms-wmv</td>
<td>Windows Media file with audio and/or video content. Typically used to download and play files or to stream content.</td>
</tr>
<tr>
<td>.asf</td>
<td>video/x-ms-asf</td>
<td>Legacy content. Typically used to download and play files or to stream content. May contain audio and/or video content. Typically used to download and play files or to stream content.</td>
</tr>
<tr>
<td>.wm</td>
<td>video/x-ms-wm</td>
<td>Reserved</td>
</tr>
<tr>
<td>.wax</td>
<td>audio/x-ms-wax</td>
<td>Metafiles that reference Windows Media files with the .asf, .wma or .wax file extensions.</td>
</tr>
<tr>
<td>.wvx</td>
<td>video/x-ms-wvx</td>
<td>Metafiles that reference Windows Media files with the .wma, .wmv, .wvx or .wax file extensions.</td>
</tr>
<tr>
<td>.asx</td>
<td>video/x-ms-asf</td>
<td>Metafiles that reference Windows Media files with the .wma, .wax, .wmv, .wvx, .asf, or .asx file extensions.</td>
</tr>
<tr>
<td>.wmx</td>
<td>video/x-ms-wvx</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Remarks

Scripting and digital rights management (DRM) must be supported by any application rendering Windows Media files.

See Also

- Windows Media Metafile Elements Reference
Windows Media Player

The Windows Media Player SDK provides features that affect the behavior of Windows Media Player.

The following sections detail SDK features that apply to Windows Media Player in general.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registry Settings</td>
<td>Details values that you can change in the user's registry to enable Windows Media Player to recognize custom file name extensions.</td>
</tr>
<tr>
<td>Command Line Parameters</td>
<td>Details the set of command line parameters that specify how the Player behaves when it starts.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player 9 Series SDK

Registry Settings

Windows Media Player 9 Series uses the registry to store certain settings. You can make changes to these settings to change the behavior of Windows Media Player and the Player control.
The following sections detail the supported registry settings.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name Extension Registry Settings</td>
<td>Describes how to register a custom file name extension.</td>
</tr>
<tr>
<td>Custom Scheme Registry Settings</td>
<td>Describes how to register a custom scheme.</td>
</tr>
<tr>
<td>Folder Monitoring Registry Settings</td>
<td>Describes how to register file folders to be monitored by the Player.</td>
</tr>
</tbody>
</table>

See Also

- Windows Media Player

File Name Extension Registry Settings

Windows Media Player maintains a list of file name extensions in the registry on the user's computer. When the user attempts to play a digital media file from the local computer or from a computer on a corporate intranet by using the Universal Naming Convention (UNC), the Player checks the file name extension against the list in the registry. If a match is found, the Player then checks two values that indicate permissions for the file type (such as playback or add to Media Library) and which underlying technology, or runtime (such as Microsoft DirectShow® or the Windows Media Format SDK), can be used to play the file. If no match is found, the Player presents the user with a warning dialog box that prompts the user for permission to attempt to play the file. If you create digital media files with custom file name extensions, you can prevent this warning from appearing on the user's computer by registering the file name extension and providing values for permissions and runtime.

The list of file name extensions is maintained as a set of registry keys that match the registered file name extensions, including the dot (.) separator. A separate list is maintained for the local machine and for each user. For the local machine, the file name extension keys are subkeys of the following registry key:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Multimedia\WMPlayer\Extensions\
```

For example, the key for files having a .xyz file name extension for the local machine would be:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Multimedia\WMPlayer\Extensions\xyz
```

To change values in this key or to create a new subkey, the current user must be an administrator for the
For individual users, the file name extension keys are subkeys of the following registry key:

**HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Player\Extensions\**

For example, the key for files having a .xyz file name extension for the current user would be:

**HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Player\Extensions\.xyz**

When checking for registered file name extensions, the Player first checks the values located in **HKEY_LOCAL_MACHINE**. If none are found for the current file name extension, the Player next checks the values in **HKEY_CURRENT_USER**. If none are found for the current file name extension, the Player displays the warning message to the user.

Each file name extension subkey may include the following two values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissions</td>
<td>REG_DWORD</td>
<td>A value that represents the operations allowed for digital media files having the registered file name extension.</td>
</tr>
<tr>
<td>Runtime</td>
<td>REG_DWORD</td>
<td>A value that represents the underlying technology that Windows Media Player can use to play digital media files having the registered file name extension.</td>
</tr>
</tbody>
</table>

The following table provides possible values for **Permissions**.

<table>
<thead>
<tr>
<th>Value (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Permission for playback. Files having the registered file name extension can be played.</td>
</tr>
<tr>
<td>2</td>
<td>Permission for folder drop. Files having the registered file name extension will be included in the playlist created when the user drags a folder containing the files and drops it on the Player user interface.</td>
</tr>
<tr>
<td>4</td>
<td>Permission for media CD. Files having the registered file name extension will be included in the playlist created when a CD containing the files is inserted into the CD-ROM drive.</td>
</tr>
<tr>
<td>8</td>
<td>Permission for <strong>Media Library</strong>. Files having the registered file name extension can be added to <strong>Media Library</strong>.</td>
</tr>
<tr>
<td>16</td>
<td>Permission for HTML streaming. Files having the registered file name extension will be inserted into the Internet Explorer cache when delivered from a</td>
</tr>
</tbody>
</table>
The values in the preceding table may be combined. For instance, a value of "0x003" means permission for both playback and folder drop.

Failing to set a value of 1 for **Permissions** (permission for playback) for a file name extension that Windows Media Player registers by default will have no effect. The Player will always grant playback permission for files having a file name extension that is registered by default.

Setting a value of 2 for **Permissions** (permission for folder drop) for a file name extension other than one that Windows Media Player registers by default will have no effect. You can, however, remove this permission from a default file name extension to prevent files having that file name extension from being included in the playlist during a drag-and-drop operation.

The following table provides possible values for **Runtime**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Render using the Windows Media Format SDK.</td>
</tr>
<tr>
<td>7</td>
<td>Render using Microsoft DirectShow.</td>
</tr>
</tbody>
</table>

Changing the **Runtime** value for a file name extension that Windows Media Player registers by default will have no effect. The Player will always use the default **Runtime** value for files having a file name extension that is registered by default. This registry value is designed to allow runtime configuration for custom file name extensions.

**See Also**

- Specifying Media Library Classification
- Registry Settings

---

*Note* Use of this feature requires Windows Media Player QFE 823275.

If you specify permission for **Media Library** (8) for your custom file name extension, you can add a REG_SZ registry value that specifies whether your files appear in **Media Library** in the **All Music** or **All Video** node. To do this, create a new value in the following registry key:
HKEY_LOCAL_MACHINE\Software\Microsoft\MediaPlayer\MLS\Extensions

For example, the following registry syntax specifies that files having the file name extension ".xyz" will appear in the All Music node in Media Library:

[HKEY_LOCAL_MACHINE\Software\Microsoft\MediaPlayer\MLS\Extensions]
"xyz" = "audio"

The following table provides possible values for custom file name extensions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>Custom file types appear in the All Music node in Media Library.</td>
</tr>
<tr>
<td>video</td>
<td>Custom file types appear in the All Video node in Media Library.</td>
</tr>
</tbody>
</table>

To change values in this key, the current user must be an administrator for the computer.

If you do not specify a value for your custom file name extension, your files will appear in the Other Media node by default.

See Also

- File Name Extension Registry Settings

Custom Scheme Registry Settings

Schemes are custom protocols. Windows Media Player maintains a list of schemes in the registry on the user's computer. When the user attempts to play a digital media file, the Player first checks whether the Windows Media Format SDK supports the scheme. If it doesn't, the Player checks the scheme against the list in the registry. If a match is found, the Player then checks a value that indicates which underlying technology, or runtime (such as Microsoft DirectShow or the Windows Media Format SDK), can be used to play the file. If no match is found, the Player presents the user with a warning dialog box that prompts the user for permission to attempt to play the file. If you stream digital media files using a custom protocol scheme, you can prevent this warning from appearing on the user's computer by registering the scheme and providing a value for the runtime.

The list of schemes is maintained as a set of registry keys that match the registered schemes, without the colon and the two slashes (://). For example, the key for the wmlhtml:// scheme, which is used to stream rich media,
named "wmhtml". A separate list is maintained for the local machine and for each user. For the local machine, the scheme keys are subkeys of the following registry key:

HKEY_LOCAL_MACHINE\Microsoft\Multimedia\WMPlayer\Schemes\

For example, the key for the wmhtml:// scheme for the local machine is:

HKEY_LOCAL_MACHINE\Microsoft\Multimedia\WMPlayer\Schemes\wmhtml

To change values in this key or to create a new subkey, the current user must be an administrator for the computer.

For individual users, the scheme keys are subkeys of the following registry key:

HKEY_CURRENT_USER\Microsoft\MediaPlayer\Player\Schemes\

For example, the key for the wmhtml:// scheme for the current user is:

HKEY_CURRENT_USER\Microsoft\MediaPlayer\Player\Schemes\wmhtml

When checking for registered schemes, the Player first checks the values located in HKEY_LOCAL_MACHINE. If none are found for the current scheme, the Player next checks the values in HKEY_CURRENT_USER. If none are found for the current scheme, the Player displays the warning to the user.

Each scheme subkey may contain one of the following possible values for Runtime.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Render using the Windows Media Format SDK.</td>
</tr>
<tr>
<td>7</td>
<td>Render using Microsoft DirectShow.</td>
</tr>
</tbody>
</table>

Changing the Runtime value for a scheme that the Windows Media Format SDK supports will have no effect. The Player will always use the Format SDK as the runtime for schemes that the Format SDK supports. This registry value is designed to allow runtime configuration for custom schemes.

See Also

- Registry Settings
Folder Monitoring Registry Settings

Windows Media Player 9 Series can monitor file folders to check whether new digital media files have been added. When the Player detects a new file in a monitored folder, it automatically adds the file to Media Library. Users can change the list of monitored folders by clicking Monitor Folders on the Media Library tab of the Options dialog box. You programmatically add a folder for monitoring by adding a value to the registry.

To add a folder for monitoring you must first create or modify two values in the following registry key:

HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Preferences\n
The new values must be named as follows.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrackFoldersDirectories</td>
<td>REG_DWORD</td>
<td>A DWORD value that represents the count of folders to monitor. This must match the count of TrackFoldersDirectories(X) values.</td>
</tr>
<tr>
<td>TrackFoldersDirectories(X), where (X) represents an integer value.</td>
<td>REG_SZ</td>
<td>A string value that represents the path of the folder to monitor. Each folder to monitor requires a separate value. The suffix ((X)) should always begin at 0 and then increment by 1. Windows Media Player also monitors subfolders of the specified folder.</td>
</tr>
</tbody>
</table>

For example, to add the first folder to monitor, add the following value:

```
[HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Preferences]
"TrackFoldersDirectories0" = "c:\MyPath\MyFolder"
```

Then, create the count value:

```
[HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Preferences]
"TrackFoldersDirectories" = dword:00000001
```

To add a second folder to monitor, add the following value:

```
[HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Preferences]
"TrackFoldersDirectories1" = "c:\MyPath\MyFolder2"
```

Then, increment the count value:

```
[HKEY_CURRENT_USER\Software\Microsoft\MediaPlayer\Preferences]
"TrackFoldersDirectories" = dword:00000002
```

See Also

- Registry Settings
Command Line Parameters

Windows Media Player 9 Series supports a set of command line parameters that specify how the Player behaves when it starts. The following table details the parameters and their behaviors.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;path\filename&quot;</td>
<td>Start the Player and play the file.</td>
</tr>
<tr>
<td>(For example: <code>wmplayer &quot;c:\filename.wma&quot;</code>)</td>
<td></td>
</tr>
<tr>
<td>&quot;path\filename&quot; /fullscreen</td>
<td>Play the specified file in full-screen mode.</td>
</tr>
<tr>
<td>(For example: <code>wmplayer &quot;c:\filename.wmv&quot; /fullscreen</code>)</td>
<td></td>
</tr>
<tr>
<td>/Device: {DVD</td>
<td>AudioCD}</td>
</tr>
<tr>
<td>(For example: <code>wmplayer /device:audio CD</code>)</td>
<td></td>
</tr>
<tr>
<td>&quot;path\filename&quot;?WMPSkin=skin name</td>
<td>Open the Player, applying the specified skin.</td>
</tr>
<tr>
<td>For example: <code>wmplayer &quot;c:\filename.wma&quot;?wmpskin=headspace</code></td>
<td></td>
</tr>
<tr>
<td>/Task NowPlaying</td>
<td>Open the Player in the Now Playing feature.</td>
</tr>
<tr>
<td>/Task MediaGuide</td>
<td>Open the Player in the Media Guide feature.</td>
</tr>
<tr>
<td>/Task CDAudio</td>
<td>Open the Player in the Copy from CD feature.</td>
</tr>
<tr>
<td>/Task MediaLibrary</td>
<td>Open the Player in the Media Library feature.</td>
</tr>
<tr>
<td>/Task RadioTuner</td>
<td>Open the Player in the Radio Tuner feature.</td>
</tr>
<tr>
<td>/Task PortableDevice</td>
<td>Open the Player in the Copy to CD or Device feature.</td>
</tr>
<tr>
<td>/Task Services /Service servicename</td>
<td>Open the Player in the Premium Services feature, showing the service specified by the servicename parameter. This value is the unique name for the service.</td>
</tr>
</tbody>
</table>
See Also

- Windows Media Player

Glossary

To find a term in the glossary, click the letter of the alphabet that is the first letter in the term you want to look up.

You can also read glossary terms within the text of Help by clicking the underlined glossary term links. After you click a glossary link, the glossary term and definition appear in a pop-up window. To close the window, click anywhere on the screen.

A  B  C  D  E  F  G  H  I  J  K  L  M
N  O  P  Q  R  S  T  U  V  W  X  Y  Z

ambient

Supported by more than one skin element.

analog

The traditional format in which audio and video are transmitted by using a wave or analog signal. An analog signal may not work with digital speakers; computers use digital signals.

anchor window

A small window that can appear in the lower-right corner of the screen when Windows Media Player is in skin mode. You can use the window to return to full mode and access other commands.

announcement
A Windows Media metafile that gives a player the information needed to receive content. Announcement files contain Extensible Markup Language (XML) scripts.

**aspect ratio**

The ratio of the width of an image to its height.

**attribute**

A name-value data pair.

**bandwidth**

A network's capacity for transferring an amount of data in a given time.

**bit rate**

The number of bits transferred per second.

**broadcast**

A method by which a client receives a stream. During a broadcast connection, clients cannot control the stream. This is the opposite of an on-demand presentation.

**broadcast unicast**

A point-to-point connection that a client initiates to a publishing point in a Windows Media server.

**buffer**

An area of computer memory reserved for temporarily holding data before that data is used on the receiving computer. Buffering protects against the interruption of data flow.

**caption**

Text that accompanies images or videos, either as a supplemental description or a transcript of spoken words.

**CD**

*See definition for: compact disc (CD)*

**client**
Any computer or program connecting to, or requesting the services of, another computer or program. Client can also refer to the software that enables the computer or program to establish the connection.

**clipping image**

An image that is not displayed, but instead defines the visible region of another image.

**codec**

An abbreviation for compressor/decompressor. Software or hardware used to compress and decompress digital media.

**compact disc (CD)**

An optical storage medium for digital data.

**compression**

A process for removing redundant data from a digital media file or stream to reduce its size or the bandwidth used.

**content**

Audio, video, images, text, or any other information that is contained in a digital media file or stream.

**download**

To transfer a file over a network in response to a request from the device that receives the data. Downloaded content is kept on the receiving device for playback on demand. In contrast, streamed content is played as it is delivered.

**element**

A fundamental syntactic unit in markup languages, such as HTML or XML. Elements are delimited by start tags and end tags. Empty elements are defined using an empty-element tag.

**encode**

To convert audio and video content to a specified digital format.
error correction

In Windows Media Player, a process to ensure that digital audio data is read from the CD-ROM drive accurately during playback or copying. Using error correction can prevent undesirable noises that are not part of the original material.

Extensible Markup Language (XML)

A markup language that provides a format for describing structured data. XML is a World Wide Web Consortium (W3C) specification, and is a subset of Standard Generalized Markup Language (SGML).

file format

The structure or organization of data in a file. File format is usually indicated by the file name extension.

See also: file name extension, file type

file name extension

A set of characters added to the end of a file name that identifies the file type or format.

See also: file format, file type

file type

A description of the content or format of a file. File type is usually indicated by the file name extension.

See also: file format, file name extension

frame rate

The number of video frames displayed per second. Higher frame rates generally produce smoother movement in the picture.

full mode

The default operational state of Windows Media Player in which all of its features are displayed. The Player can also appear in skin mode.

See also: skin mode

G

global attribute
An attribute associated with a skin definition file as a whole, which can be accessed from any element.

**H**

**hover image**

An image that is displayed for a control whenever the mouse pointer hovers over it.

**HTTP**

*See definition for:* [Hypertext Transfer Protocol (HTTP)]

**Hypertext Transfer Protocol (HTTP)**

The Internet protocol used to deliver information over the World Wide Web.

**I**

**intelligent streaming**

A type of streaming that detects network conditions and adjusts the properties of a video or audio stream to maximize quality.

**L**

**licensed file**

A Windows Media file that has an associated license that defines how the file can be played. The restrictions stated in the license vary depending on the license creator. When a CD track is copied by using Windows Media Player, a license can be assigned to the newly created file. Under that license, the file can only be played on the computer where the file was created.

**lossy compression**

A process for compressing data in which information deemed unnecessary is removed and cannot be recovered upon decompression. Typically used with audio and visual data in which a slight degradation of quality is acceptable.

**M**

**mapping image**

An image that is not displayed, but instead defines significant regions on another image through the use of
metadata

Data about data. Title, subject, author, and size are examples of a file's metadata.

metafile playlist

A Windows Media metafile that contains information about a series of digital media items.

Microsoft Media Server (MMS) protocol

A proprietary protocol using UDP or TCP to deliver content as a unicast stream.

See also: Transmission Control Protocol (TCP), User Datagram Protocol (UDP)

MIDI

See definition for: Musical Instrument Digital Interface (MIDI)

MIME

See definition for: Multipurpose Internet Mail Extension (MIME)

MMS protocol

See definition for: Microsoft Media Server (MMS) protocol

Moving Picture Experts Group (MPEG)

The committee that creates international standards for coding audio-visual information to a digital, compressed format. The acronym MPEG is appended to the beginning of individual specifications developed by the committee. For example, MPEG-2 refers to the standard, ISO/IEC - 11172.

MPEG

See definition for: Moving Picture Experts Group (MPEG)

multicast

A content delivery method in which a single stream is transmitted from a media server to multiple clients. The clients have no connection with the server. Instead, the server sends a single copy of the stream across the network to multicast-enabled routers, which replicate the data. Clients can then receive the stream by monitoring a specific multicast IP address and port.

Multipurpose Internet Mail Extension (MIME)

A standard that extends the Simple Mail Transport Protocol (SMTP) for encoding non-ASCII data files such as video, sound, and binary files for attachment to Internet e-mail.

Musical Instrument Digital Interface (MIDI)
A specification of the MIDI Manufacturers Association (MMA). The specification defines a protocol for describing music data, such as note on and note off messages; a file format for storing music data, called Standard MIDI; and a standard hardware interface.

Packet
A unit of information transmitted as a whole from one device to another on a network.

Playlist
A list of digital media content.

Port
A connection point in a computer through which a peripheral device or another computer can communicate.

Progress bar
A meter-like indicator within a graphical user interface that illustrates the progress of an incremental process, such as a file download.

Protocol
A set of formats and procedures that enable computers to exchange information.

See also: Hypertext Transfer Protocol (HTTP), Microsoft Media Server (MMS) protocol, Transmission Control Protocol (TCP), User Datagram Protocol (UDP)

Proxy server
A server located on a network between client software, such as a Web browser, and another server. It intercepts all requests to the server to determine whether it can fulfill them itself. If not, it forwards the request to another server.

Publishing point
An organized memory location that is identified by a name on a Windows Media server. The name is part of the URL used by a client when requesting content from the server. See also: broadcast publishing point, on-demand publishing point.

Sami
See definition for: Synchronized Accessible Media Interchange (SAMI)
**skin**

A user interface that provides an alternative appearance and customized functionality for software such as Windows Media Player.

**skin definition file**

An XML document that specifies the elements in a skin, along with their relationships and functionality. A skin definition file has a .wms file name extension.

**skin mode**

An operational state of Windows Media Player in which its user interface is displayed as a skin.

*See also:* [full mode](#), [skin](#)

**stream**

Digital media that is in the process of being delivered in a continuous flow across a network.

**streaming**

A method of delivering digital media across a network in a continuous flow. The digital media is played by client software as it is received. Typically, streaming makes it unnecessary for users to download a file before playing it.

**Synchronized Accessible Media Interchange (SAMI)**

An XML-based language for specifying closed captions in multiple languages and styles.

*See also:* [caption](#)

**TCP**

*See definition for:* Transmission Control Protocol (TCP)

**track**

An individual song or other discrete piece of audio content.

**Transmission Control Protocol (TCP)**

The protocol within TCP/IP that governs the breakup of data messages into packets to be sent via IP, and the reassembly and verification of the complete messages from packets received by IP.
Uniform Resource Locator (URL)

The address of a resource on a network. It conforms to the protocol://servername/resource syntax, where protocol is the name of the protocol used (such as http), servername is the name of the server, and resource is the directory path and file name of the resource. If a resource is not specified, a default one is often provided by the server.

URL

See definition for: Uniform Resource Locator (URL)

User Datagram Protocol (UDP)

A connectionless transport protocol in the TCP/IP protocol stack that is used in cases where some packet loss is acceptable, for example, with digital media streams.

W

Windows Media client

The ActiveX control called Windows Media Player that receives and renders content from Windows Media server components. The client can be on either the same computer as the server, or another computer.

Windows Media file

A file that contains audio, video, or script data. The content of the file is encoded with one of the Windows Media codecs.

Windows Media Format

The format of a digital media file or stream that was encoded with Windows Media codecs.

Windows Media metafile

In Windows Media technologies, a file that provides information about Windows Media files and their presentation. File name extensions for Windows Media metafiles include .asx, .wax, .wvx, .wmx, and .nsc.
See definition for: Extensible Markup Language (XML)

Back to Top

Previous

© 2000-2003 Microsoft Corporation. All rights reserved.