Getting Started

This section introduces you to the DirectShow SDK and helps you get oriented. It includes a description of the SDK, information about porting from previous versions, background needed to understand different parts of DirectShow, a guide to the documentation, and answers to frequently asked questions.

What is DirectShow?

The Microsoft® DirectShow™ SDK gives developers access to DirectShow services, which provide playback multimedia streams from local files or Internet servers, and capture of multimedia streams from devices. Specifically, this enables playback of video and audio content compressed in various formats, including MPEG, Apple® QuickTime®, audio-video interleaved (AVI), and WAV, and both Video for Windows-based capture and WDM-based (Windows Driver Model) capture.

At the heart of the DirectShow services is a modular system of pluggable components called filters, arranged in a configuration called a filter graph. A component called the filter graph manager oversees the connection of these filters and controls the stream's data flow.

Applications control the filter graph's activities by communicating with the filter graph manager. You can do this indirectly by using the ActiveMovie Control, or directly by calling COM interface methods. The SDK also enables you to create your own filters using the DirectShow class library. The base classes in the C/C++ library implement the required COM interfaces on the filters and provide the basic filter framework.

The DirectShow SDK is based on several other Microsoft services and supports at least one other Microsoft service. For example, the video display components in DirectShow rely on Microsoft® DirectX® services whenever possible.

The amount you must know about an underlying or supported technology depends on what you are doing. For example, you'll need to understand COM programming if you are using C or C++ to control DirectShow playback or create a filter. But you don't need to understand COM programming to use the ActiveMovie Control.

DirectShow developers can use the following Microsoft services (you can find information about most of these in the Platform SDK or on the World Wide Web at http://www.microsoft.com/msdn/.)

- DirectX
- Component Object Model (COM)
- Media control interface (MCI)

You can also use the following Microsoft languages for DirectShow development:

- Microsoft Visual Basic® — used to create and control filter graphs.
- Microsoft Visual C++® — used to create and control filter graphs, and create filters.

See the product documentation for more information on these languages.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

What's New in the DirectShow SDK?

The Microsoft® DirectShow™ SDK (formerly the Microsoft ActiveMovie™ SDK) is now part of the Microsoft DirectX® Media SDK. Features added since ActiveMovie 1.*x* include video and audio capture and compression, device enumeration, digital versatile disc (DVD) support, digital video (DV) support, cutlist support, device control and timecode support, and multimedia streaming. See the topics in the following lists for details of all the DirectShow SDK's new features. For the most recent updates to this documentation, consult the Microsoft DirectX Web site at http://www.microsoft.com/DirectX/.

Articles

- Porting Code from ActiveMovie
- What Background Do You Need?
- Documentation Roadmap
- About Capture Filter Graphs

- Improving Capture Performance
- Creating a Capture Application
- Write an Audio Capture Filter
- Write a Video Capture Filter
- Enumerate and Access Hardware Devices in DirectShow Applications
- DVD for Title Vendors
- DirectShow DVD Support
- About Cutlists
- Using Cutlists
- About Compression Filters
- Recompress an AVI File
- DV Data in the AVI File Format
- AVI 2.0 File Format Extensions
- Play a Movie in a Window Using DirectDrawEx and Multimedia Streaming
- Play a Movie from C++
- Control the Video Playback Window from C++
- Display a Filter's Property Page from C++
- Use Multimedia Streaming in DirectShow Applications
- Build a Filter or Application with Visual C++ 5.x
- Exposing Capture and Compression Formats
- About the DirectShow Filter Graph Editor
- Using the Filter Graph Editor
- About WDM Video Capture
- Using DirectDrawEx

Multimedia Streaming

- About the Multimedia Streaming Architecture
- List of Multimedia Streaming Interfaces
- Multimedia Streaming Reference
- Multimedia Streaming Data Types
- Error and Success Codes for Multimedia Streaming
- Multimedia Streaming Component Objects
- Multimedia Streaming Sample Code

DirectDrawEx

- What Is DirectDrawEx?
- IDirectDrawFactory Interface
- IDirectDraw3 Interface

DirectShow COM Interfaces

The following lists group the new DirectShow interfaces by area. In addition, the Multimedia Streaming Reference contains interfaces specific to multimedia streaming.

Capture, Compression, Device Enumeration, and Windows Driver Model (WDM) Capture Interfaces

- IAMAudioInputMixer

- IAMBufferNegotiation
- IAMCrossbar
- IAMDroppedFrames
- IAMStreamConfig
- IAMStreamControl
- IAMStreamSelect
- IAMTVTuner
- IAMVfwCaptureDialogs
- IAMVfwCompressDialogs
- IAMVideoCompression
- IAMVideoProcAmp
- ICaptureGraphBuilder
- IConfigAviMux
- IConfigInterleaving
- ICreateDevEnum
- IFileSinkFilter2
- IFilterGraph2
- IMediaPropertyBag
- ISeekingPassThru

DirectSound Interface

- IAMDirectSound

Digital Versatile Disc (DVD), Closed Captioning, Property Set, and Video Port (VP) Interfaces

- IAMLine21Decoder
- IDvdControl
- IDvdGraphBuilder
- IDvdInfo
- IKsPropertySet
- IVPBaseConfig
- IVPBaseNotify
- IVPConfig
- IVPNotify

Cutlist Interfaces

- IAMAudioCutListElement
- IAMCutListElement
- ICutListGraphBuilder
- IFileClip
- IAMFileCutListElement
- IStandardCutList
- IAMVideoCutListElement

Device Control and Timecode Interfaces

- IAMExtDevice

- IAMExtTransport
- IAMTimecodeDisplay
- IAMTimecodeGenerator
- IAMTimecodeReader

Memory Allocation/Media Sample Interfaces

- IAMDevMemoryAllocator
- IAMDevMemoryControl
- IMediaSample2

Samples

- AMCap Sample (DirectShow Capture Application)
- CLText Sample (Text Cutlist Application)
- ShowStrm Sample (Multimedia Streaming Application)
- Simplecl Sample (Cutlist Application)
- Vcrctrl Sample (VCR Control Filter)
- VidClip Sample (Video Editing Application)
- VidCap Sample (Video Capture Filter)
- Dvdsampl Sample (DVD Player Application)
- PlayFile Sample (Simple Playback Application)
- InWindow Sample (Window Playback Application)
- MPEGProp Sample (MPEG Property Page Display Application)

Filters

The following lists group the filters by category. Some filters can fit in more than one category so be sure to look at the different categories to see what new filters are available.

Capture, WDM Capture, and File Writing Filters

- Analog Video Crossbar
- Audio Capture
- AVI MUX
- File Writer
- TV Audio
- TV Tuner
- VFW Video Capture
- WDM Video Capture

Digital Video (DV) Filters

- DV Muxer
- DV Splitter
- DV Video Decoder
- DV Video Encoder

Cutlist Filter

- Cutlist File Source

Closed Captioning Filters

- Lyric Parser
- Line 21 Decoder
- Multi-File Parser
- SAMI (CC) Parser

Compression and Decompression Filters (Codecs)

- ACM Audio Compressor
- AVI Compressor
- Indeo 4.3 Video Compression
- Indeo 4.3 Video Decompression
- Indeo 5.0 Audio Decompression
- Indeo 5.0 Video Compression
- Indeo 5.0 Video Decompression
- QuickTime Decompressor
- TrueMotion 2.0 Decompressor

Parser, Renderer, and Mixer Filters

- AVI Draw
- DSound Audio Renderer
- File Stream Renderer
- Internal Script Command Renderer
- MIDI Parser
- MIDI Renderer
- Overlay Mixer

DVD Filter

- DVD Navigator

Classes, Functions, and Macros

- CBaseRenderer callback function class
- CBaseStreamControl class
- Functions for creating DLLs and registering and unregistering filters
- IUnknown Macro

The following material relating to the ActiveMovie Control has been significantly updated:

- About the DirectShow ActiveMovie Control
- Control Events
- Control Methods
- Control Properties
- Control Property Sheet

Porting Code from ActiveMovie

This article describes the steps you must take to port your Microsoft® ActiveMovie™ 1.0 code to Microsoft DirectShow™.

Contents of this article:

Recompiling ActiveMovie 1.0 Code with DirectShow

If you are recompiling your ActiveMovie 1.0 code with the DirectShow header files and libraries, you must make the following changes:

1. The name of the **IFilter** interface has been changed to IBaseFilter. You need to change the name to compile with DirectShow libraries and headers. ActiveMovie 1.0 binaries will continue to work because the IID (Interface Identifier) GUID for the interface is still the same.
2. The constructor of the CBasePropertyPage::CBasePropertyPage function has one less parameter. Once again, if you are recompiling your code with the DirectShow base class libraries, you will have to modify the parameters for the CBasePropertyPage constructor.

   The old prototype for the constructor was:

```
CBasePropertyPage(TCHAR *pName,       // Debug only name
                      LPUNKNOWN pUnk,         // COM Delegator
                      HRESULT *phr,               // Return code
                      int DialogId,                   // Resource ID
                      int TitleId);                     // To get title
```

   The new prototype is:

```
CBasePropertyPage(TCHAR *pName,       // Debug only name
                      LPUNKNOWN pUnk,         // COM Delegator
                      int DialogId,                   // Resource ID
                      int TitleId);                     // To get title
```

Interfaces and Services Improved Since ActiveMovie 1.0

Some interfaces and services in ActiveMovie 1.0 have been improved in DirectShow. You should use the following improved interfaces and services:

1. A new interface, IFilterMapper2, has been introduced to support the concept of filter categories and registering filters in correct categories. IFilterMapper will still work for filters that were shipped with ActiveMovie 1.0 and are now part of the standard filters in the Filter Graph Editor. **IFilterMapper** might not do the right thing for filters that are in the new categories. For example, you might end up with filters that you can't instantiate.
2. The AMovieSetupRegisterFilter2 function has been added to use **IFilterMapper2**. AMovieSetupRegisterFilter will still work and will use IFilterMapper.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

What Background Do You Need?

The amount you need to know for a particular DirectShow task depends on the task. For example, you must understand basic COM principles and C or C++ to create a filter. But you don't need to understand either to use the ActiveMovie Control.

The following tables show what you might need to know to perform different tasks and get the most out of the DirectShow documentation. To find the information you need, see Documentation Roadmap for a description of what you'll find in the different sections of the DirectShow documentation. See Where Can I Learn About... for the location of answers to specific questions within the DirectShow documentation, and see How to Get More Information for the location of useful information outside the DirectShow documentation.

The following table shows the background you might need to understand different sections of the DirectShow documentation.

Section	Background needed
Getting Started	None.
Using the ActiveMovie Control	You must know Microsoft® Visual Basic® or C/C++ if you're using the control from within them. If you're using the control as an end-user and not developing applications with it, the section is self-sufficient.
DirectShow Basics	None.
Application Developer's Guide	You must know the language you are developing your application in, and basic COM programming.
Filter Developer's Guide	You must be experienced with C/C++ and basic COM programming.

C/C++ Reference	You must know C/C++ and basic COM. For the "Debugging" section, it would be useful to understand basic debugging procedures, such as how to generate debug information or check for memory leaks. For the Event Notification Codes section, it might be helpful to know a little about Windows messaging.
Filters and Samples	You must know C/C++ and basic COM. Depending on your filter, you might need to know a little about DirectSound® and DirectDraw®, or understand multimedia capture and compression.
Appendixes	You must know a little about C to understand Reserved Identifiers, and you need to know a little about multimedia to understand Media Types.
Glossary	None.

The following table shows the background you might need to accomplish a sampling of tasks in DirectShow.

Task	**Background needed**
Use the ActiveMovie Control as an end-user.	None.
Use the Filter Graph Editor tool.	None.
Write a filter.	You must be experienced with C/C++ and basic COM.
Create or control a filter graph in Visual Basic.	You must be familiar with Visual Basic and basic COM.
Play a movie.	To play a movie in a Visual Basic or C/C++ application, you must be familiar with those languages and with basic COM programming. To play a movie from a Web page, you should be familiar with HTML, but the DirectShow documents will explain the rudiments, such as using the OBJECT and HREF tags.
Write a media-streaming application.	You must be familiar with C/C++ and basic COM.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Documentation Roadmap

The Microsoft® DirectShow™ SDK documentation is delivered in HTML format (which you can read by using a browser) and in .ivt format (which you can read using Microsoft InfoViewer). The Microsoft Developer Network (MSDN) Library and Microsoft Visual C++®, as well as other Microsoft products, install InfoViewer for you.

When you install the DirectShow SDK, the HTML documentation is always installed. The HTML start page, Start.htm, is available in the main installation directory (Dxmedia by default). If the DirectShow installation finds InfoViewer on your computer, it also installs and registers the .ivt documentation. The next time you open InfoViewer (through Visual C++, MSDN, or another program) the DirectShow SDK documentation will be available under the "DirectX Media SDK" heading.

For the most recent updates to this documentation, consult the Microsoft DirectX Web site at http://www.microsoft.com/DirectX/.

The DirectShow SDK documentation contains general sections that provide information useful to everyone, and sections that apply to particular tasks. Getting Started and DirectShow Basics contain high-level information useful to everyone. ActiveMovie Control, Application Developer's Guide, and Filter's Developer's Guide contain procedures and technical information specific to their respective topics. Multimedia Streaming describes the simplified streaming interfaces that enable developers to interface to a stream without creating a custom renderer or source filter. DirectDrawEx describes the DirectDrawEx dynamic-link library, which extends the current functionality of DirectDraw. Filters and Samples contains a brief description of the filters and sample applications shipped with DirectShow. C/C++ Reference is the DirectShow Reference. Appendixes contain supplemental information to the reference, and Glossary defines DirectShow terms.

See What Background Do You Need? to find out what background different tasks require.

Contents of this article:

To help you find the information you need, the following list describes the content of each section in the DirectShow documentation and when you will typically use it.

- Getting Started — gives general information about DirectShow, such as background needed to use DirectShow, material new in this release, and frequently asked questions. Read this section to orient yourself when first starting with DirectShow.
- ActiveMovie Control — describes the ActiveMovie Control's features and how to use it to play back movies in different applications, such as C/C++, Visual Basic, or an HTML Web page. This section also contains the HTML reference and the Visual Basic 5.*x* reference, which describes using DirectShow COM interfaces as Visual Basic objects. Read this section if you want to use the control.
- DirectShow Basics — contains overview articles covering basic DirectShow concepts, such as filter graph architecture and data flow, how to use the Filter Graph Editor tool, and a list of the filters and sample filters supplied with DirectShow. Read this section for a high-level introduction to DirectShow, whether you are developing your own filters or developing an application that uses the supplied filters.
- Application Developer's Guide — contains articles about developing applications, including a discussion of media streaming, a list of application interfaces, and step-by-step procedures for adding features to your applications. Read this section if you want to develop applications with DirectShow.
- Filter's Developer's Guide — contains technical articles about creating filters, including different filter types, connecting and controlling filters, and step-by-step procedures for creating specific filters, such as capture filters and DirectSound renderer filters. Read this section if you want to write your own filters.
- C/C++ Reference — contains the COM interface and class library references, structures,

utility functions, events, error messages, and debugging tips. Read this section to get details about an interface or a data type.
- Filters and Samples — contains a brief description of the filters and sample applications shipped with DirectShow and available for use or for modification in your own applications. Read this section to create a prototype application from the supplied filters or to see if the filter you want to create already exists.
- Multimedia Streaming — contains an overview of the multimedia streaming architecture, a list of the multimedia streaming interfaces, and the multimedia interface references. Read this section to learn how to use the multimedia streaming interfaces to automatically negotiate the transfer and conversion of data from the source to the application, so you don't have to write code to handle the connection, transfer of data, data conversion, and actual data rendering or file storage. The streaming interfaces provide a uniform and predictable method of data access and control, which makes it easy for an application to play back data, regardless of its original source or format.
- DirectDrawEx — contains an overview of the DirectDrawEx dynamic-link library, which extends the current functionality of DirectDraw, and reference material for the IDirectDrawFactory and IDirectDraw3 interfaces.
- Appendixes — contains miscellaneous technical information, including a list of AM_MEDIA_TYPE structure values supported in DirectShow, information about time stamps and DirectShow DVD support, country codes and channel-to-frequency mappings for TV tuner applications, and the reserved identifiers in the DirectShow header files. Read this section if you need details about one of these topics.
- Glossary — defines DirectShow-specific terminology you might not be familiar with.

Where Can I Learn About...

This section tells you where to find information in the DirectShow documentation about specific tasks and topics. Also see the "Frequently Asked Questions" section for answers to common questions.

Q. Where should I begin reading in the DirectShow documentation?

A. Read for an overview and then read the articles in the and sections that match your interest. See Debugging with DirectShow for debugging information.

Q: Where can I find information about capture?

A: See the following **capture topics** .

- About Capture Filter Graphs
- Improving Capture Performance
- Creating a Capture Application
- Recompress an AVI File
- Write an Audio Capture Filter
- Write a Video Capture Filter
- Enumerate and Access Hardware Devices in DirectShow Applications

DirectShow provides the following capture-related filters. The video renderer is often used for purposes other than capture.

- ACM Audio Compressor
- Audio Capture

- AVI MUX
- DV Muxer filter
- File Writer
- VFW Video Capture
- VidCap Sample (Video Capture Filter)
- Video Renderer

Capture-related interfaces:

- IAMAudioInputMixer
- IAMDroppedFrames
- IAMStreamConfig
- IAMVfwCaptureDialogs
- IAMVfwCompressDialogs
- IAMVideoCompression
- ICaptureGraphBuilder
- ICreateDevEnum

Q. Where can I find information about playing back movies using the C or C++ language?

A. Read Play a Movie from C++ in the .

Q. Where can I find information about playing back movies using Microsoft® Visual Basic®?

A. If you are using the ActiveMovie Control, see Using the ActiveMovie Control in Visual Basic. If you are constructing or controlling a filter graph with Visual Basic, see Constructing Filter Graphs Using Visual Basic and Controlling Filter Graphs Using Visual Basic in the section.

Q. Where can I find information about playing back movies from a Web page?

A. Read Using the ActiveMovie Control in HTML Pages in the section.

Q. Where can I find a class hierarchy diagram for the DirectShow base classes?

A. Each class in the DirectShow C++ Class Library section has its own class hierarchy diagram at the top of its opening page.

Q. Where can I find information about the filters shipped with DirectShow?

A. For a description of each of the filters included with the DirectShow run time, as well as the sample filters included with the SDK, see . When the SDK is installed, all sample filters are built and registered.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

How to Get More Information

You can find most of the information you need beyond a knowledge of your particular programming language and the DirectShow documents themselves in the Platform SDK, available in the Microsoft Developer Network. See http://www.microsoft.com/msdn/ for more information.

COM Overview and DirectShow and COM can give you an introduction to Component Object Model (COM), and this might suffice for many development tasks in DirectShow. If you need more information, see the "COM" section in the Platform SDK, or an introductory book such as *Understanding ActiveX OLE* by David Chappell.

For more information about DirectDraw or DirectSound, see the Microsoft DirectX® SDK documentation in the Platform SDK.

For more information about multimedia in general, see the "Graphics and Multimedia Services" section in the Platform SDK.

For more information about Microsoft Windows® messaging, see the "Win32 Messages" section in the "Reference" portion of the Platform SDK.

For more information about media control interface (MCI), see the "MCI" section in the Platform SDK.

For more information about C/C++ programming, read the product documentation for your C/C++ compiler (such as the Microsoft Visual C++® documentation), or a standard C or C++ programming book. For more information about Visual Basic programming, read the Visual Basic product documentation.

Previous Home Topic Contents Index Next

Frequently Asked Questions

This section answers many frequently asked questions about DirectShow. It is divided into three parts: answers to general questions, answers to questions asked by application developers, and answers to questions asked by filter developers.

Contents of this article:

- Filter Development Questions

General Questions

Q. What are the differences between DirectShow™, DirectDraw®, DirectSound®, and DirectX®?

A. For an overview of these Microsoft® technologies as well as many others, see http://www.microsoft.com/directx/.

Q. What operating systems does DirectShow support?

A. DirectShow supports Windows® 95 or later and Windows NT® version 4.*x* or later.

Q. Is there an DirectShow Hardware Compatibility List (HCL)?

A. No. DirectShow uses all DirectDraw® and DirectSound hardware capabilities where they are available. Where no special hardware is available, DirectShow uses GDI to draw video and the waveOut* Multimedia APIs to play back audio.

Q. What multimedia file formats does DirectShow support?

A. DirectShow supports the following formats.

- MIDI (.mid)
- MPEG-1 (.mpg, .mpeg, .mpv, .mp2, .mpa, .mpe)
- Audio-video interleaved (.avi)
- Nonproprietary Apple® QuickTime® files (.mov, .qt)
- Wave (.wav)
- AU (.au, .snd)
- AIFF (.aif, .aifc, .aiff)

Q. Where can I obtain detailed file format specifications?

A. One source is *Encyclopedia of Graphics File Formats*, second edition, by James D. Murray and William vanRyper, published by O'Reilly & Associates, Inc. That book describes MPEG-1, AVI, and some QuickTime file formats. See AVI 2.0 File Format Extensions for more information about DirectShow's support of this format.

Q. Does DirectShow provide video capture services?

A. Yes, DirectShow provides video capture.

Q. What version of Internet Explorer do I need to use DirectShow for Web content?

A. DirectShow is designed for Internet Explorer 3.*x* and later.

Q. What HTML tags would I use with DirectShow?

A. For information on the HTML tags to use to play movies using DirectShow, see Using the ActiveMovie Control in HTML Pages.

Q. Is there any sample code showing how to program DirectShow?

A. The DirectShow SDK includes sample source code in C, C++, and Microsoft Visual Basic®. For more information, see DirectShow Samples.

Q. What compiler do I need for DirectShow development?

A. DirectShow was designed with Visual C++® 5.*x* in mind, but any compiler capable of generating Component Object Model (COM) objects should work once the compiler's environment has been configured correctly. The base class libraries might need to be rebuilt to work completely since compilers can vary between versions.

Q. Do I need a compiler to play back movies?

A. No. Once DirectShow is installed, double-click any media file to view it. If you want more specialized applications, you can program DirectShow by using either C/C++ (in which case a compiler is required) or any Automation-compatible language (such as Visual Basic).

Q. When will DirectShow be integrated with Microsoft's operating systems?

A. DirectShow will be included in the next and future versions of Windows.

Q. Is the source for the ActiveMovie Control (Amovie.ocx) available?

A. No, the source is not available.

Q. What DirectX technologies are available on Windows NT?

A. Windows NT 5.*x* supports DirectX Foundation 5.*x*, including DirectDraw, DirectPlay®, DirectSound, Direct3D®, and DirectInput®. Windows NT 4.*x* supports DirectDraw, DirectPlay, and DirectSound.

Q. Can you recommend any reference books for Windows programming or COM?

A. Several books from Microsoft Press, including *Advanced Windows* by Jeffrey Richter and *Understanding ActiveX and OLE* by David Chappell, make great references for DirectShow developers.

Q. What is a GUID?

A. A globally unique identifier (GUID) is a 128-bit (16-byte) integer that an algorithm creates. The algorithm uses several criteria, including the current date, time, and a machine identifier, to ensure that it will be unique. GUIDs are used extensively in the Component Object Model (COM) and have an important role in DirectShow.

Q. How do I get a GUID?

A. GUIDs can be generated using Guidgen.exe. Guidgen, a Windows-native program, is included with Microsoft's Visual C++® products. Developers can also use Uuidgen.exe, a console application, in the Platform SDK.

Q. When do I need to call QueryInterface?

A. Whenever you need to obtain an interface for an object.

Q. What are some typical "getting started" problems with COM?

A. Some typical problems involving COM are:

- Forgetting to create a new GUID is a very common problem in filter development. For example, if you create a new filter based on one of the sample filters, you must create a new GUID for your filter. Otherwise, when you install your filter, it will overwrite the information that was already registered for the sample filter. Create a new GUID by using the Guidgen.exe tool.
- Omitting the address of (&)' operator in the last parameter of a call to QueryInterface or CoCreateInstance causes an exception when trying to use the object specified in that last parameter. The problem can be difficult to debug, because the void cast required prevents a compiler warning. The following example shows both the bad and the good syntax in a call to **QueryInterface**.

```
IUnknown *pUnk;

// bad -- you will hit an exception when you try to use pUnk
// HRESULT hr = pObject->QueryInterface(IID_IPersist,
//                                       (void **)pUnk); // <-- bad

// good -- note the '&' is required for the last parameter
HRESULT hr = pObject->QueryInterface(IID_IPersist,
                                      (void **)&pUnk); // <-- good
```

Redistribution Questions

Q. What parts of DirectShow can I redistribute?

A. The SDK includes a redistributable package under the Redist directory.

Q. Is the source to the Filter Graph Editor (Graphedt.exe) tool available? Can the Filter Graph Editor be redistributed?

A. No, the source is not available, and Graphedt.exe is not redistributable.

Q. How can I install the DirectX Media redistributables package from my application?

A. The DirectX Media redistributable package is called Dxmedia.exe. An application must register itself as a client of the redist package by running Dxmedia.exe with the **-id** switch and a unique identifier for the application, as shown in the following syntax:

```
dxmedia.exe -id:identifier
```

The *identifier* should uniquely identify your application. It can be the GUID of your registered application, or a unique string. You should probably not use a string such as "game," but a more distinctive string. For example:

```
dxmedia.exe -id:myrocketgame
```

— or —

```
dxmedia.exe -id:my_GUID
```

The identifier is required. If you simply type "Dxmedia.exe" or simply double-click on the Dxmedia.exe icon, nothing will be installed.

The same syntax is used for all platforms, Windows 95, Windows NT x86, and Windows NT Alpha. Note that each processor has its own executable (either an Alpha or an x86 version).

If successful, the installation can return one of the following success codes:

- ERROR_SUCCESS — the installation completed successfully.
- ERROR_SUCCESS_REBOOT_REQUIRED — the installation was successful, but changes will not be effective until the system is rebooted.

If unsuccessful, the installation returns an HRESULT describing the error; for example, E_FAIL or E_INVALIDARG.

The installation is totally quiet, and no dialog boxes appear.

Q. Does the install/uninstall identifier need to be a GUID?

A. No, but it must be unique. GUIDs are essentially guaranteed to be unique.

Q. Does the install package handle different versions during install?

A. Yes, automatically. Files are only overwritten (after backup) if they use the same platform and are older than the files about to be installed.

Q. How does an application uninstall the DirectX Media package it installed during its own installation?

A. The redist package includes an uninstall executable called Purgedxm.exe. Uninstall by running Purgedxm.exe with the application's identifier that was used during install, as shown in the following syntax:

```
purgedxm.exe identifier
```

Note: Do not use the **-id** switch when uninstalling.

Q. Does the redistributable DirectX Media run-time setup install DirectX?

A. It installs the DirectX Media run-time components. It also installs a minimum version of DirectX Foundation on x86 and Alpha platforms, as appropriate. The application can install DirectX Foundation separately if the full installation is needed, after the redistributable package has been installed.

Application Development Questions

Q. How can you detect whether DirectShow is installed on a given machine?

A. Use the following code fragment to detect whether DirectShow is installed. It assumes that you have already included the Streams.h header file and initialized the COM subsystem by using the **CoInitialize** function with a null parameter. It also assumes that you will uninitialize COM by using the **CoUninitialize** function before closing your application.

```
IGraphBuilder * lpAMovie;

HRESULT hr = CoCreateInstance(
        CLSID_FilterGraph,
        0,
        CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder,
        (void **) &lpAMovie );


if (SUCCEEDED(hr)) {
    lpAMovie->Release();
    // DirectShow is installed
} else {
    // DirectShow is not installed
}
```

Q. Which interfaces do applications typically use?

A. IGraphBuilder, IMediaEvent, IMediaControl, and IVideoWindow.

Q. How do I change the owner of the video window?

A. Use the IVideoWindow::put_Owner method.

Q. How do I play a movie in a specific window?

A. Use IVideoWindow::put_Owner to specify the window you want. Set the style of the video window to include WS_CHILD using the IVideoWindow::put_WindowStyle method, and then position the video window inside your window using the IVideoWindow::SetWindowPosition method. The following sample code demonstrates this process.

```
#include <windows.h>
#include <streams.h>

#define FILENAME L"C:\\WIN95.AVI"

void PlayVideoInWindow(HWND hTargetWindow)

{   // PlayVideoInWindow //

    CoInitialize(NULL);

    HRESULT hr;
    IGraphBuilder *pigb;

    // Create an empty filter graph object
    hr = CoCreateInstance(CLSID_FilterGraph,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder,
        (void **)&pigb);

    if (FAILED(hr))
        return;

    hr = pigb->RenderFile(FILENAME, NULL);

    if (FAILED(hr)) {
        pigb->Release();
        return;
        }   // Bail out, file probably wasn't found! //

    RECT rc;
    IVideoWindow *pivw;

    hr = pigb->QueryInterface(IID_IVideoWindow, (void **)&pivw);
    pivw->put_Owner((OAHWND)hTargetWindow);

    // Here's the key: we must set the required flags,
    //   AND set the position
    pivw->put_WindowStyle(WS_CHILD | WS_CLIPCHILDREN | WS_CLIPSIBLINGS);
    GetClientRect(hTargetWindow, &rc);
    pivw->SetWindowPosition(rc.left, rc.top, rc.right rc.bottom);

    IMediaControl *pimc;

    hr = pigb->QueryInterface(IID_IMediaControl, (void **)&pimc);

    long l;
    IMediaEvent *pime;

    hr = pigb->QueryInterface(IID_IMediaEvent, (void **)&pime);
    pimc->Run();

    pime->WaitForCompletion(INFINITE, &l);

    // The following MUST be called otherwise nasty things can happen!
    pivw->put_Owner(NULL);

    pime->Release();
    pimc->Release();
    pivw->Release();
    pigb->Release();

    CoUninitialize();

}   // PlayVideoInWindow
```

Q. Why doesn't my video always repaint correctly after being covered by another window?

A. You need to specify the WS_CLIPCHILDREN style for the video window's owner.

Q. When I try to build an application using DirectShow I get the following link error:

```
Error LNK2001: unresolved external symbol IID_IGraphBuilder
```

I have included the Strfim.h file provided with the DirectX Media SDK but within

Strfim.h IDD_IGraphBuilder is defined as an external. Why?

A. The symbols are defined in Strmiids.lib, which is one of the libraries located in the \LIB directory. You need to link this library into your project. See Build a Filter or Application with Visual C++ 5.x for information on how to do this in Microsoft Developer Studio. The VC5KIT kit in the \TOOLS directory also describes how to configure Developer Studio to build DirectShow-based applications.

Q. Where are the DirectShow-specific return codes (HRESULTs) **defined?**

A. The error return codes specific to DirectShow are found in Vfwmsgs.h, located in the DirectShow SDK Include directory. More general errors, such as error codes returned by CoCreateInstance, can be found in \Msdev\Include\Winerror.h if you have Microsoft Developer Studio. Vfwmsgs.h and Winerror.h also provide information about the layout of HRESULT values. Before you search for a particular error in Winerror.h, you might need to convert the low portion to decimal, depending on the error.

Q. How do I interpret errors from Visual Basic?

A. Calls to DirectShow object methods and properties can return error codes; these codes are stored in (and can be retrieved from) the Number property of the Err object. Error numbers can be returned in one of two forms: an DirectShow error or a Visual Basic run-time error code. DirectShow errors are referred to by number, starting at 0x80040200 (2147746304 in decimal). You can look up the error number by its hexadecimal value in the Include\Vfwmsgs.h header file. When Visual Basic can interpret the external object error, it generates a Visual Basic run-time error. For example, the E_INVALIDARG return code from an interface generates Visual Basic run-time error 5, "remote procedure call failed". In cases when Visual Basic cannot interpret a system error (such as E_ABORT), error 287 is returned in the **Err** object.

Q. Why does the filter graph manager return E_NOTIMPL when I call IVideoWindow **methods?**

A. The E_NOTIMPL return value indicates that no filter is in the graph that supports **IVideoWindow**. In other words, your graph does not contain a video renderer. One solution is to call the IGraphBuilder::Render method and allow the filter graph manager to automatically insert any necessary filters and complete the filter graph.

Q. Why does the filter graph manager return VFW_E_NOT_CONNECTED when I call IVideoWindow **methods?**

A. The VFW_E_NOT_CONNECTED return value indicates that the video renderer's input pin is not connected.

Q. I want to use the ActiveMovie Control to play a video, but I only want the image to show and not the toolbar or controls. How do I do this?

A. You can turn off the controls and display through the property pages. Or, you can change the options programmatically by setting the ShowControls and ShowDisplay properties to FALSE (0).

Q. How do enumerators work?

A. Enumerators are COM objects created to traverse an ordered set. The application or filter can call a method such as IEnumPins::Next to obtain an item from the set. Enumerators are usually matched to the data type they retrieve. In DirectShow, there are different enumerators to retrieve items such as filters, pins, and media types.

Q. How do I change a filter's settings without displaying the property page?

A. If the filter exposes a custom interface for this purpose, then you can access the filter properties. The DirectShow SDK includes a sample called Contrast, which exposes methods such as get_ContrastLevel and put_ContrastLevel to enable you to change its properties programmatically. For more details, see the implementation of Contrast's property page. You can find a description of the Contrast filter in About Effect Filters and Contrast Sample (Video Contrast Filter).

Q. Can I change the properties of the MPEG audio decoder without displaying its property page?

A. Currently the MPEG audio codec does not allow you to set its properties programmatically.

Q. Can DirectShow notify me of its current position on a regular basis?

A. There is no callback notification of position. Use a timer and poll for the current position by using the IMediaSeeking::GetCurrentPosition method.

Q. My application monitors the frame rate and sets a new frame rate every time a certain event takes place. However, playback stutters as the rate adjusts. How do I switch the rate smoothly?

A. Alter the time stamps instead of changing the frame rate.

Filter Development Questions

Q. What programming languages can filters be developed in?

A. You can write DirectShow filters in any language that can generate objects adhering to Microsoft's Component Object Model (COM). The base classes for DirectShow are written in C++.

Q. Can I develop my filter using the Microsoft Foundation Classes (MFC)?

A.The DirectShow class library is totally independent of MFC and contains most of the base classes you might need for filter development. See Build a Filter or Application with Visual C++ 5.x for information on how to build DirectShow applications with Visual C++. You can use the VC5Kit located in the Tools directory for assistance in building these applications.

Q. What are the differences between source, rendering, and transform filters?

A. Source filters form the point of origin for data, while rendering filters present the data in a final format on various devices such as a video card, audio card, disk file, and so on. Transform filters manipulate, modify, or alter the data in some way.

Q. How do I install my filter?

A. For information about how to install (self-register) a filter, see Register DirectShow Objects and the IAMovieSetup interface.

Q. Where do I install my filter on an end-user system?

A. You can install the filter anywhere you want, because the filter registration process records the filter's full path and file name at the time of registration. There is no need to put your filter in the Windows System or System32 directory.

Q. What causes the Filter Graph Editor to report that "The filter could not be created. Resources used by this filter might already be in use." when I try to insert a filter?

A. The Filter Graph Editor (Graphedt.exe) displays this message when it can't find the filter's .ax file. The filter has probably been moved, renamed, deleted, or was not properly set up. It needs to be properly registered. For information about how to self-register a filter, see Register DirectShow Objects and the IAMovieSetup interface.

Q. What is the resolution of time stamps in DirectShow?

A. The minimum resolution is 100 nanoseconds.

Q. Can a filter graph have only one filter that performs the work of source, transform, and renderer?

A. Yes, but to take advantage of DirectShow's "plug-in" nature, it is best to use different combinations of filters chained together. This also provides easier code reuse and flexibility in design, and is strongly recommended.

Q. Can I have a filter with many input and output pins (with some fixed one-to-one correspondence between them)?

A. Yes. Ensure that you implement the IPin::QueryInternalConnections method on every pin. Without this, the graph builder will assume that each input pin streams through to every output pin. While you can have a filter with many input and output pins, it's often more desirable to have several filters. Otherwise, you might find yourself writing more code (overriding more and more functions from the base classes) than if you just use an individual filter for each stream.

Q. Why does the graph builder stream each of my input pins through to every output pin?

A. This is the default behavior. Implement the IPin::QueryInternalConnections method on every pin to avoid this.

Q. Can I use a filter outside an DirectShow filter graph?

A. No, filters are integrated with the DirectShow architecture and require a filter graph.

Q. How do I determine the number of pins on a filter?

A. Call the IBaseFilter::EnumPins method, and then keep calling the IEnumPins::Next method until you don't get any more pins.

Q. At what privilege ring do filters run?

A. Filters in DirectShow run at user-privilege level three (ring three on x86 processors).

Q. How do filters communicate with one another?

A. Filters typically communicate with one another through their pins. Pins negotiate a common format and transport for exchanging data.

Q. Can I test my filter with the Filter Graph Editor (Graphedt.exe)?

A. Graphedt was designed to help filter developers visualize the connections and interaction between filters during filter development. It is not meant as a robust test platform.

Q. Can pin direction change dynamically?

A. DirectShow pins are not designed to have their direction changed. You could create two pins, one for each direction, and use a single piece of code to drive both pins.

Q. Why is the MPEG audio codec filter sometimes missing from an auto-rendered graph? How can the MPEG stream splitter audio output pin connect directly to an audio renderer?

A. Codec filters are only required by the renderer if the audio driver does not support a given audio format. Renderers often query the audio driver inside **CheckMediaType** to determine if the proposed format is supported, and if supported, pass the media samples directly to the audio subsystem when the graph is run. If the driver incorrectly indicates a format is supported, the most obvious symptom is garbled audio. You then need to get corrected drivers from your audio board manufacturer.

Q. My source/capture filter feeds streaming data from a network card into a filter graph. How do I get the DirectShow ActiveMovie Control to automatically use my source filter?

A1. If you want your source filter to be recognized like the DirectShow file and URL source filters, then use one of the methods below.

Method 1: Refer to your object using some protocol name you make up (for example, ABC). Add an entry in the registry under HKEY_CLASSES_ROOT\ABC, specifying the CLSID of your source filter. Your source filter will be used for your fictional ABC protocol, much like HTTP is used for internet addresses.

Method 2: Use a special file to start up your filter with a known set of bytes (for example a UUID you define yourself). Add the necessary check bytes to the registry under HKEY_CLASSES_ROOT\Media Type, making up your own major type, or stick with MEDIATYPE_Stream as other filters do and specify the proper subtype. Put your filter's CLSID in the Source Filter value. Check bytes are (decimal start, decimal length, mask (default all FF), hex value) so for a CLSID at the start you want (0, 36, , your CLSID). See Registering a Custom File Type for more information.

A2. If you want your capture filter to show up in the list of devices in a video capture application, see Write a Video Capture Filter.

Previous Home Topic Contents Index Next

ActiveMovie Control

The ActiveMovie™ Control enables you to add movies and sound to your applications and Web pages. You can play back multimedia files that are in formats such as MPEG, AVI, WAV, and MOV, among others. Microsoft® Internet Explorer and the DirectShow SDK include the ActiveMovie Control. The DirectShow SDK provides background information about DirectShow's capabilities and architecture beyond the information provided in this section.

Using the ActiveMovie Control

This section explains how to use the Microsoft® ActiveMovie™ Control, both in HTML pages and in Visual Basic applications. It contains a description of the control properties, methods, events, property sheet, and shortcut keys.

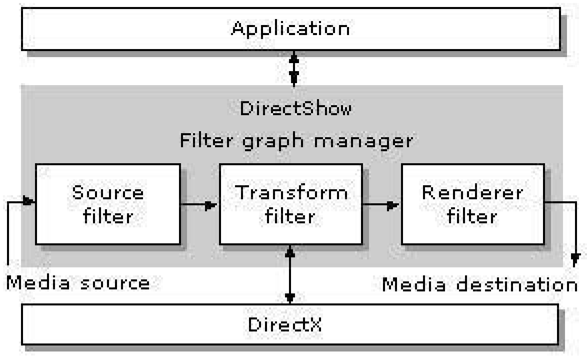
Previous Home Topic Contents Index Next

Introduction to DirectShow

Microsoft® DirectShow™ is an extensible media architecture that delivers high-quality audio and video playback from the Internet or an intranet. DirectShow supports the most popular media types, including MPEG audio and video, AVI video, WAV audio, MIDI audio, and Apple® QuickTime® video. You can access the DirectShow-supported media types quickly and easily by using the Microsoft ActiveMovie™ Control.

The DirectShow architecture defines how applications can control and process time-stamped multimedia data using modular components called *filters* connected in a configuration called a *filter graph*. A complete filter graph consists of a number of filters, assembled in a logical progression from the data source to the media renderer or renderers.

Applications assemble the filter graph and control how data moves through it by accessing the *filter graph manager* through programming interfaces, as shown in the following illustration.



For example, the Microsoft MPEG filter graph uses the following filters.

- A source filter to read the data off the disk
- A splitter transform filter to separate the video and audio
- A video transform filter to decompress the video data
- A video rendering filter to display the data on the screen
- An audio transform filter to decompress the audio data
- An audio rendering filter to send the audio data to the sound card

Default filter graphs are configured for you when you install the DirectShow software on your computer. You can also install additional filters and create your own filter graphs. For more information about creating and managing filter graphs, see Filter Graph Manager and Filter Graphs in the DirectShow SDK documentation.

About the DirectShow ActiveMovie Control

You can use the DirectShow ActiveMovie Control to quickly add support for multimedia to your applications. DirectShow supports the many media types, including MPEG audio and video, AVI video, WAV audio, MIDI audio, and Apple® QuickTime® video. The DirectShow SDK provides samples and examples that illustrate how to write programs that use the control with either Microsoft® Visual Basic® or Microsoft Visual C++®.

The ActiveMovie Control represents an easy-to-use programming interface that lets you manage multimedia using the control's properties, methods, and events. The control handles all video and audio rendering for you, which simplifies your programming tasks and makes it easy to add support for multimedia to your application. In addition to the ActiveMovie Control, you can use the DirectShow Component Object Model (COM) interfaces. (See Summary of DirectShow COM Interfaces in the DirectShow SDK documentation.)

Using the ActiveMovie Control in HTML Pages

Files on a Web page are referenced using URL names and various protocols, such as HTTP, file, FTP, or Gopher. These names can be either explicit or relative.

Following are two examples of explicit references. The first example uses the HTTP protocol to access the file. The second uses a network path and file name.

"http://directshow/samples/web/RoadRun.avi"

"file://directshow/samples/web/RoadRun.avi"

Here is an example of a relative reference:

```
"/samples/web/RoadRun.avi"
```

In the examples in this article, you can assume that either relative or explicit file names will work and any protocol for accessing these file names will also work.

Contents of this article:

Invoking Playback on HTML Pages

You can play files in one of two ways: either embedded in a Web page, or externally in a window displayed outside or on top of the page. There are four ways of playing back media using HTML tags. Three of these result in embedded playback:

- OBJECT
- EMBED
- IMG DYNSRC=

The fourth means of implementing playback results in an external window:

- A HREF

Although not restricted to playback, you can use an attribute of the OBJECT tag to automatically download a file that contains required components for playback.

- OBJECT CODEBASE

The following examples show how you use these tags to reference the file, but do not show all possible properties and attributes (width, height, color, and so on) that could be specified as well. See Setting ActiveMovie Control Properties on a Web Page for more information.

Using the OBJECT Tag

When you use the OBJECT tag, you must explicitly specify the class identifier (CLSID) for the playback control, as shown in the following example.

```
<OBJECT CLASSID="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A">
<PARAM NAME="FileName" VALUE="//directshow/samples/web/RoadRun.avi">
</OBJECT>
```

The preceding CLSID is that of the ActiveMovie Control. The application loads the control and passes it the file name. The control parses the file name and downloads and plays the file. If the protocol is "file," the control instantiates the File Source (async) filter to play back the file. If the protocol is "http," the control instantiates the File Source (URL) filter instead. See the DirectShow SDK documentation for more information on these filters.

Using this tag, you can play back certain media types using "progressive downloading," which allows the user to play back the downloaded portion of the file as the data is downloading. See How Progressive Downloading Works for more information.

Using this tag along with the CODEBASE attribute, you can download components required for playback. See Using the OBJECT Tag with the CODEBASE Attribute for more information.

Using the OBJECT Tag with the CODEBASE Attribute

As the author of a Web page, you might find that using components such as specialized DirectShow filters can enhance playback of your media files on the user's machine. However, if the necessary components aren't already installed on the user's computer, you need a method to install those components. Internet Explorer supports using the CODEBASE attribute of the OBJECT tag for automatic file download, and cabinet (.cab) files for automatic installation. See http://www.microsoft.com/workshop/prog/cab/default.htm for information about the Cabinet SDK and .cab files.

The CODEBASE attribute works in conjunction with the OBJECT tag. See Using the OBJECT Tag for another example that uses the OBJECT tag.

An example of filter download using the CODEBASE attribute and a .cab file follows:

```
<OBJECT ID=GargleFilter1
CLASSID="CLSID:d616f350-d622-11ce-aac5-0020af0b99a3"
CODEBASE="http://directshow/samples/gargle.cab">
</OBJECT>
```

The preceding class identifier (CLSID) is the ID of the Gargle filter, which applies a "gargle" sound effect to a sound file. Internet Explorer checks the target computer's registry for the specified CLSID. If the specified CLSID is not present (indicating the filter is not installed on the machine), the browser attempts to find the file (Gargle.cab) at the site specified by the CODEBASE attribute and download it. Once downloaded, the .cab file provides the information to automatically install the filter.

Note: The .cab file must have been properly constructed for automatic installation to work. The .cab must include the filter (Gargle.ax in this case) and an .inf file that contains installation information. The .inf file should handle copying each filter to a specific directory (such as the Windows® System directory) on the user's computer and properly registering each filter. See the Platform SDK for information about .inf files.

Using the EMBED Tag

Netscape introduced this tag for embedding source. The following example uses this tag.

```
<EMBED autostart="FALSE" loop="FALSE" SRC="http://directshow/samples/web/RoadRun.av
```

This tag works identically to the OBJECT tag, except that you don't need to specify the CLSID. Internally, DirectShow examines the HKCR/MIME/Database/Content Type registry entries, retrieves the appropriate CLSID (DirectShow automatically registers all compatible data types at install time), and launches the control with that media type.

Using the IMG tag with the DYNSRC= Attribute

Internet Explorer can use the DYNSRC= attribute to play back audio-video interleaved (AVI) files. You can use this playback mechanism, but it is preferable to use the OBJECT tag. The following example demonstrates this HTML tag.

```
<IMG start=1 loop=0 DYNSRC="http://directshow/samples/web/RoadRun.avi">
```

Using the A Tag with the HREF Attribute

To play back a movie in an external movie, use the A tag with the HREF attribute. An example of this follows:

```
<A HREF="http://directshow/samples/web/dwad.cinepack.avi">AVI_CINEPACK</A>
```

The parsing steps that Internet Explorer 3.0 (or later) goes through are similar to the "EMBED SRC tag" case, in that it gets the file extension, maps it to a content type and looks in the HKCR/MIME/Database/Content Type area of the registry to get a CLSID to invoke. However, in this case the control plays in an external window. The ActiveMovie Control is in control of downloading the file.

How Progressive Downloading Works

The following series of steps describes what happens during progressive downloading.

1. As soon as possible after activation, the ActiveMovie Control displays the first frame in the file (for movies). The filter graph remains in a paused state.
2. The control does not start to play until it determines that it has enough data to play uninterrupted while the remaining amount downloads. At this time, playback starts and downloading will continue.
3. The user can click the Play button during this time. In this case, the control will play until the play cursor reaches the end of the current amount downloaded and will automatically transition from the running state to the paused state.
4. The download of the file will not stop if the user clicks Stop. The user can stop the download by closing the control.
5. For most AVI or QuickTime files, you can't start playback until DirectShow has read the entire file (because the indexes are at the end of the file). However, the download bar is displayed to show how much of the file has been downloaded. The Play control button remains unavailable until the control downloads the file completely.

Setting ActiveMovie Control Properties on a Web Page

You can set any ActiveMovie Control property by using the PARAM tag inside an OBJECT container. For Boolean values, −1 is TRUE and 0 is FALSE. All other values are set as they would be in the Microsoft® Visual Basic® design environment.

The following example demonstrates playing a movie with most controls shown. If you have a video file and you want to try this, you can copy and paste this code onto your Web page and modify the parameters as you want. If you do this, be sure to change the movie's file name, perhaps using a relative path and file protocol (for example, PARAM NAME="FileName" VALUE="file://c:\mymovie.mpg") to play the movie from your hard drive.

```
<HTML>
<HEAD>
<TITLE>ActiveMovie Embedded MPG Object Test Page</TITLE>
</HEAD>
<BODY>
Scene from Stargate (MPG File)

<OBJECT ID="ActiveMovie1" WIDTH=357 HEIGHT=322
CLASSID="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A">
```

```
<PARAM NAME="Version" VALUE="1">
<PARAM NAME="EnableContextMenu" VALUE="-1">
<PARAM NAME="ShowDisplay" VALUE="-1">
<PARAM NAME="ShowControls" VALUE="-1">
<PARAM NAME="ShowPositionControls" VALUE="0">
<PARAM NAME="ShowSelectionControls" VALUE="0">
<PARAM NAME="EnablePositionControls" VALUE="-1">
<PARAM NAME="EnableSelectionControls" VALUE="-1">
<PARAM NAME="ShowTracker" VALUE="-1">
<PARAM NAME="EnableTracker" VALUE="-1">
<PARAM NAME="AllowHideDisplay" VALUE="-1">
<PARAM NAME="AllowHideControls" VALUE="-1">
<PARAM NAME="MovieWindowSize" VALUE="0">
<PARAM NAME="FullScreenMode" VALUE="0">
<PARAM NAME="MovieWindowWidth" VALUE="353">
<PARAM NAME="MovieWindowHeight" VALUE="318">
<PARAM NAME="AutoStart" VALUE="0">
<PARAM NAME="AutoRewind" VALUE="-1">
<PARAM NAME="PlayCount" VALUE="1">
<PARAM NAME="SelectionStart" VALUE="0">
<PARAM NAME="SelectionEnd" VALUE="48.5151388">
<PARAM NAME="Appearance" VALUE="1">
<PARAM NAME="BorderStyle" VALUE="1">
<PARAM NAME="FileName" VALUE="http://DirectShow/samples/web/stargate1.mpg">
<PARAM NAME="DisplayMode" VALUE="0">
<PARAM NAME="AllowChangeDisplayMode" VALUE="-1">
<PARAM NAME="DisplayForeColor" VALUE="16777215">
<PARAM NAME="DisplayBackColor" VALUE="0">
</OBJECT>
</BODY>
```

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Using the ActiveMovie Control in Visual Basic

This article describes how Microsoft® Visual Basic® applications can use the ActiveMovie Control. The ActiveMovie Control is a high-level interface that meets the needs of most multimedia application developers. Additional lower-level interfaces are also available to Visual Basic programmers.

Contents of this article:

Quick Start: Insert and Use the ActiveMovie Control

This section describes how to insert the ActiveMovie Control and use it in a Visual Basic application. It is a simple example to get you started, and implements only playing and stopping a movie.

Follow these steps to insert the ActiveMovie Control into a Visual Basic form and play a movie:

1. Install Internet Explorer to install the ActiveMovie Control.
2. Open a project or create a new project in Visual Basic.
3. Choose **Components** from the **Projects** menu. Choose the **Controls** tab. If the ActiveMovie Control does not appear in the list of **Controls**, click the **Browse** button. In the **Add ActiveX Control** dialog box that appears, navigate to the directory containing the control (by default, the \Windows\System or \Winnt\System32 directory). Double-click the Amovie.ocx file to add the ActiveMovie Control to the **Controls** list.
4. Add the ActiveMovie Control to the Visual Basic toolbox by selecting the check box next to **Microsoft ActiveMovie Control** in the **Controls** list.
5. Add the ActiveMovie Control to your form and size the control to the screen size you want for your movie. (Click the control in the Toolbox, and then draw its size on the form.)
6. In the **Properties** list for the ActiveMovie Control, initialize the **FileName** property to the movie file you want to play. The **FileName** property should contain the full path. For example:

   ```
   c:\movies\mymovie.avi
   ```

7. Set the **ShowControls** and **ShowDisplay** properties to FALSE.
8. Add code to start and stop the movie to two events the ActiveMovie Control supports. For example:

   ```
   Private Sub ActiveMovie1_Click()
       ActiveMovie1.Run
   End Sub

   Private Sub ActiveMovie1_KeyPress(KeyAscii As Integer)
        ActiveMovie1.Stop
   End Sub
   ```

9. Compile the application.
10. Run the application. In this example, click anywhere on the movie screen to start the movie. Press any key to stop the movie.

Follow these steps to use the ActiveMovie Control to play a movie controlled by Visual Basic controls:

1. Follow steps 1-6 as shown earlier in this section.
2. Add Visual Basic controls (such as a CommandButton or ListBox) to your form to play and stop the movie.
3. Add code to play and stop the movie. For example:

   ```
   Private Sub Command1_Click()
   ActiveMovie1.Run
   End Sub

   Private Sub Command2_Click()
   ActiveMovie1.Stop
   End Sub
   ```

4. Compile the application.
5. Run the application. In this example, Click the Command1 button to start the movie. Click the Command2 button to stop the movie.

The Visual Basic Ocxvb01 Sample

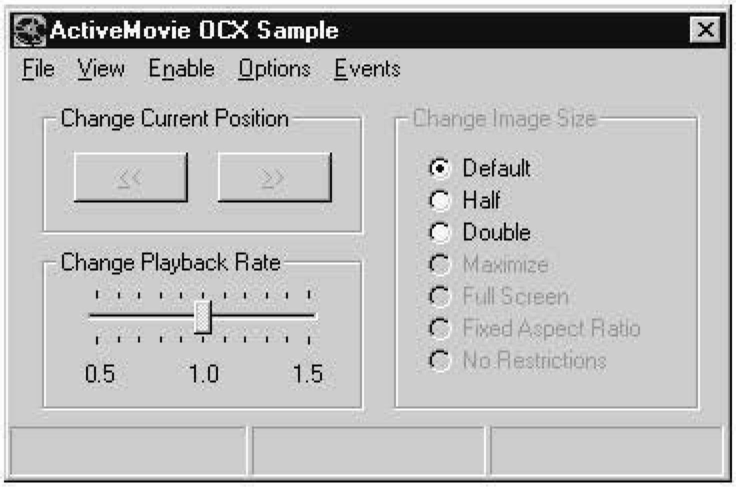
This section describes some of the features of the DirectShow Visual Basic sample application called Ocxvb01 in the VB\OCX directory of the DirectShow samples. The ActiveMovie Control manages most of the details of the display and playback of DirectShow files, while giving the Visual Basic developer control over the image size, playback rate, volume, balance, and position. With the control, the developer can manage a variety of standard user interface controls, such as rewind and fast forward, and a trackbar control to set positions within the media file.

The Visual Basic sample application Ocxvb01 consists of two forms: the main control form frmMain (Ocxvb01.frm) and the display form frmViewer (Viewer.frm).

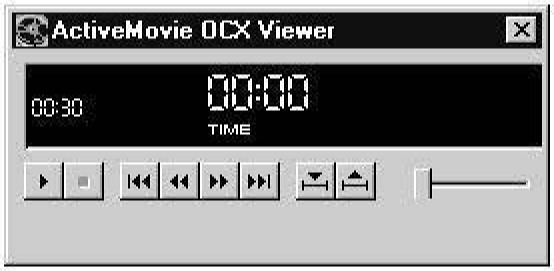
The frmMain form's menu commands call ActiveMovie Control methods to run, pause, and stop multimedia playback, set and retrieve the control's properties, and enable and disable different parts of the user interface.

The frmMain form contains two command buttons that change the CurrentPosition property of the IMediaPosition object; you can use these buttons to move forward or backward through the multimedia source file, if the multimedia source type supports this functionality. The frmMain form's trackbar control sets the Playback property; its valid range is 0.5 to 1.5. This value is used as a multiplier; 1.0 is the authored speed, so 0.5 is half the authored speed and 2.0 is twice the authored speed. You can set this property to values outside this range, but the audio portion tends to become incomprehensible.

The main control form also contains option buttons that set the display configuration to half size, full size (default), and double size. The main control form appears as shown in the following illustration.



The display form, frmViewer, contains the ActiveMovie Control that is displayed when the DirectShow multimedia source file is active or when the source is playing. The application resizes the display form to correspond to the selected source. The display form, with all ActiveMovie Control user interface elements visible and enabled, appears as shown in the following illustration.



The following sections describe how to use some of the properties, methods, and events.

Opening and Running DirectShow Source Files

To load and play a DirectShow file, set the ActiveMovie Control's FileName property. Depending on your application's requirements, you can play the file from your code by using methods like Run and Stop, or you can let the user interact with the user interface elements offered by the control.

The FileName property specifies the name of the multimedia source file. When you set the **FileName** property, several other properties are updated to indicate characteristics of that source file.

Once the FileName property contains a valid file name, you can call the control's Run method to play the multimedia file. Or, you can enable the control buttons on the display form and let the user play the file.

The sample application contains an Open command on the File menu that sets the value of the FileName property. It invokes the File Open common dialog box to obtain a file name:

```
   Private Sub mnu_File_Open_Click()

      CommonDialog1.Filter = "All files (*.*)|*.*|DirectShow files
(*.mpg;*.mpa;*.mpv;*.mov;*.mpeg;*.enc;*.m1v;*.mp2)|*.mpg;*.mpa;*.mpv;*.mov;*.mpeg;*
mp2|Audio files (.wav)|*.wav|Video for Windows files
(.avi)|*.avi"
      CommonDialog1.Flags = 4 'Hide read-only check box
      CommonDialog1.ShowOpen
      ' only set the property if the user selected a filename from the common dialc
      If CommonDialog1.filename <> "" Then
         frmViewer.ActiveMovie1.filename = CommonDialog1.filename
         g_FileOpened = True
         g_FileExtension = Right$(CommonDialog1.filename, Len(CommonDialog1.filenam
InStr(CommonDialog1.filename, "."))
      Else
         GoTo err_FileOpen
      End If
      Call ResizeViewer
      ...
```

First, the subroutine prepares the common dialog box to display only movie files. After the common dialog box returns, the subroutine checks to determine if a file name was returned.

If a file name was returned, the subroutine sets the FileName property of the ActiveMovie

Control, in addition to some other global variables: gFileOpened to determine whether or not a file has been opened, and gFileExtension to determine the type of the file.

Finally, the subroutine calls the ResizeViewer subroutine to make the dimensions of the viewer form match those of the DirectShow file. The dimensions of the Visual Basic form are expressed in twips, while the dimensions of the multimedia source are expressed in pixels. The ResizeViewer subroutine considers these differences to size the form by using the form's ScaleHeight and ScaleWidth properties:

```
' Resize form to dimensions of ActiveMovie Control + nonclient region.
    With frmViewer
        .Visible = False
        .Height = .ActiveMovie1.Height + (.Height - .ScaleHeight)
        .Width = .ActiveMovie1.Width + (.Width - .ScaleWidth)
        .Visible = True
    End With
```

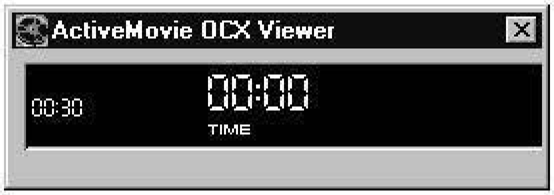
After the file is loaded successfully, you can play it by clicking Play on the control, or by using the Run method. The sample application provides a Run command on its File menu that calls the **Run** method:

```
   Private Sub mnu_File_Run_Click()

      If g_FileOpened = True Then
         frmViewer.ActiveMovie1.Run
         frmViewer.ZOrder 0
      End If
```

Managing the User Interface of the ActiveMovie Control

The display form of the sample application (Viewer.frm) contains the ActiveMovie Control. The following illustration shows the control with the status bar visible and the control bar hidden.



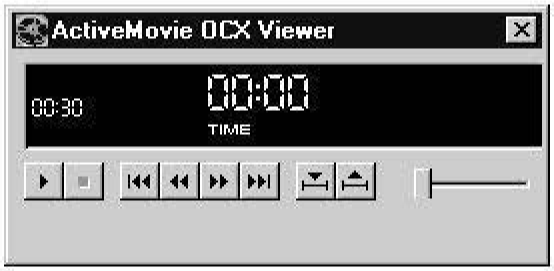
You can use the ActiveMovie Control properties to show or hide and enable or disable user interface elements of the ActiveMovie Control. The sample application includes selected menu commands that enable you to individually control each of the user interface elements. The commands on the View menu enable you to control whether the element is visible. The commands on the Enable menu enable you to control whether the element is enabled.

When the sample application loads a new file, it resets the ActiveMovie Control properties to default values and initializes the values of these selected menu commands. This fragment is from the mnu_File_Open_Click procedure:

```
   With frmViewer.ActiveMovie1
      .EnablePositionControls = False             'Disable/Enable controls.
      .EnableSelectionControls = False
      .EnableTracker = False
```

```
        .ShowPositionControls = False                   'Show/don't show controls.
        .ShowSelectionControls = False
        .ShowTracker = True
        mnu_Enable_PositionControls.Checked = False  'Check/uncheck menus to match
        mnu_Enable_SelectionControls.Checked = False 'controls.
        mnu_Enable_Tracker.Checked = False
        mnu_View_Tracker.Checked = True
        mnu_View_PositionControls.Checked = False
        mnu_View_SelectionControls.Checked = False
        ...
    End With
```

Many of the control elements have two properties associated with them: one to enable them and another to make them visible. To use the position controls, set both the EnablePositionControls and ShowPositionControls properties. The following illustration shows the display controls and all position controls with all elements of the user interface both visible and enabled.



Monitoring ActiveMovie Control Events

The ActiveMovie Control automatically monitors certain events and calls event procedures accordingly. To handle one of these events, you provide only the event handling code. For example, the StateChange event indicates a change in the state of the multimedia source, such as the change from running to stopped, or the change from paused to running. To take some action when this event occurs, the application provides code as part of the control's **StateChange** event.

To demonstrate these event handlers, the sample application provides trivial code that increments a variable. The variable represents a count of the number of times the event has occurred. The current values for these counter variables are displayed at the bottom of the main form. The following example shows the code for the StateChange event.

```
    Private Sub ActiveMovie1_StateChange(ByVal oldState As Long, ByVal newState As L

        g_cStateChange = g_cStateChange + 1
        UpdateStatusBar
```

Each event procedure passes some informational parameters too. For the StateChange event, you can determine the previous state, in addition to the control's current state, by examining the *oldState* and *newState* parameters.

Control Properties

The ActiveMovie Control supports the following properties.

Property	Description
AllowChangeDisplayMode	Indicates whether or not the end-user can change the display mode at run time between seconds and frames.
AllowHideControls	Indicates whether or not the end-user can hide the control panel at run time.
AllowHideDisplay	Indicates whether or not the end-user can hide the display panel at run time.
Appearance	Specifies the appearance of the display panel's border.
AutoRewind	Indicates whether or not the multimedia stream should automatically return to the selection's starting point when it reaches the end of the selection.
AutoStart	Indicates whether or not to automatically start playing the multimedia stream.
Balance	Specifies the stereo balance.
BorderStyle	Specifies the control's border style.
CurrentPosition	Specifies the current position within the playback file, in seconds.
CurrentState	Specifies the playback file's current state: stopped, paused, or running.
DisplayBackColor	Specifies the display panel's background color.
DisplayForeColor	Specifies the display panel's foreground color.
DisplayMode	Indicates whether or not the display panel shows the current position in seconds or frames.
EnableContextMenu	Indicates whether or not to enable the shortcut menu.
Enabled	Specifies whether or not the control is enabled.
EnablePositionControls	Indicates whether or not to show the position buttons in the controls panel.
EnableSelectionControls	Indicates whether or not to show the selection buttons in the controls panel.
EnableTracker	Indicates whether or not to show the trackbar control in the controls panel.
FileName	Specifies the name of the source data file.
FilterGraph	Contains the IUnknown interface pointer to the current filter graph object.
FilterGraphDispatch	Contains the IDispatch interface pointer to the current filter graph object.
FullScreenMode	Expands the area of the playback panel to fill the entire screen.
MovieWindowSize	Specifies the size of the playback panel.
PlayCount	Specifies the number of times to play the multimedia stream.

Rate	Specifies the playback rate for the stream.
ReadyState	Specifies the state of readiness for this ActiveMovie Control, based on how completely the source file has loaded.
SelectionEnd	Specifies the ending position in this multimedia stream, in seconds, relative to the stream's beginning.
SelectionStart	Specifies the starting position in this multimedia stream, in seconds, relative to the stream's beginning.
ShowControls	Indicates whether or not the controls panel is visible.
ShowDisplay	Indicates whether or not the display panel is visible.
ShowPositionControls	Indicates whether or not the position controls are visible.
ShowSelectionControls	Indicates whether or not the selection controls are visible.
ShowTracker	Indicates whether or not the trackbar is visible.
Volume	Specifies the volume, in hundredths of decibels.

The ActiveMovie Control supports the following properties that are common to other controls: **Appearance**, **BorderStyle**, **Enabled**, and **hWnd**. For information about these properties, see the documentation for Visual Basic.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AllowChangeDisplayMode Property

Control Properties

Indicates whether or not the end-user can change the display mode at run time between time and frames.

[*form*.]*object*.**AllowChangeDisplayMode** [= { **True** | **False** }]

Remarks

This property controls run-time access to the DisplayMode property, which specifies whether to show the multimedia file's current position in time or frames. If you set this property to True, the end-user can change how the control displays the current position by right-clicking the control window and choosing Time or Frames from the shortcut menu.

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
True	(Default) The end-user can change the type of display at run time.
False	The end-user cannot change the type of display at run time.

Data Type

Boolean

Previous Home Topic Contents Index Next

AllowHideControls Property

Control Properties

Indicates whether or not the end-user can control the visibility of the controls panel at run time.

[*form*.]*object*.**AllowHideControls** [= { **True** | **False** }]

Remarks

This property controls run-time access to the ShowControls property, which hides and displays the controls panel. If you set this property to True, the end-user can show or hide the controls panel by right-clicking the control window and choosing Controls from the shortcut menu.

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
True	(Default) The end-user can hide or show the controls panel.
False	The end-user cannot hide or show the controls panel.

Data Type

Boolean

Previous Home Topic Contents Index Next

AllowHideDisplay Property

Control Properties

Indicates whether or not the end-user can hide the display panel at run time.

[*form*.]*object*.**AllowHideDisplay** [**=** { **True** | **False** }]

Remarks

This property controls run time access to the ShowDisplay property, which hides and displays the display panel. If you set this property to True, the end-user can show or hide the display panel by right-clicking the control window and choosing Display from the shortcut menu.

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The end-user can hide or show the display panel.
False	The end-user cannot hide or show the display panel.

Data Type

Boolean

Previous Home Topic Contents Index Next

Appearance Property

Control Properties

Specifies the appearance of the display panel's border.

[*form*.]*object*.**Appearance** [**=** *long*]

Remarks

This property is identical to the standard Visual Basic **Appearance** property.

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
1	(Default) The display panel has an inset border, which gives the illusion of depth. This value is the same as 3D.
0	The display panel has no border. This value is the same as Flat.

Data Type

Long

Previous Home Topic Contents Index Next

AutoRewind Property

Control Properties

Indicates whether or not the multimedia stream should automatically return to the selection's starting point when it reaches the end of the selection.

[*form*.]*object*.**AutoRewind** [**=** { **True** | **False** }]

Remarks

If you set this property to True, DirectShow sets CurrentPosition to the position specified by SelectionStart when it reaches the position specified by SelectionEnd.

To retain the current position within the multimedia stream, set **AutoRewind** to False or use the Pause method.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
True	(Default) Reposition the multimedia stream to the start of the selection after playback stops.
False	Do not reposition the multimedia stream to the start of the selection after playback stops.

Data Type

Boolean

See Also

AutoStart

AutoStart Property

Control Properties

Indicates whether or not to automatically start playing the multimedia stream.

[*form*.]*object*.**AutoStart** [= { **True** | **False** }]

Remarks

If you set this property to False, you must make an explicit call to the Run method to play the stream.

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
True	Automatically start playing the multimedia stream.
False	(Default) Do not automatically start the multimedia stream.

Data Type

Boolean

See Also

PlayCount

Balance Property

Control Properties

Specifies the stereo balance.

[*form.*]*object*.**Balance** [= *long*]

Remarks

The value 0 indicates that the sound is balanced equally between the left and right speakers. A value of –10,000 indicates that all sound is going to the left speaker and 10,000 that all sound is going to the right speaker.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
long	Numeric expression that specifies the balance value. The number ranges from –10,000 to 10,000 and defaults to 0.

Data Type

Long

See Also

Volume Property

Previous Home Topic Contents Index Next

BorderStyle Property

Control Properties

Specifies the control's border style.

[*form.*]*object*.**BorderStyle** [= *long*]

Remarks

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
0	No border.
1	(Default) Fixed, single-line border.

Data Type

Integer (Enumerated)

CurrentPosition Property

Control Properties

Specifies the current position within the multimedia stream, in seconds.

[*form*.]*object*.**CurrentPosition** [= *double*]

Remarks

The new value must be within the range specified by the SelectionStart and SelectionEnd properties.

The current position value displayed by the control's user interface can represent either seconds or frames. The DisplayMode property determines the units shown.

Setting the **CurrentPosition** property at run time is similar to a seek operation and changes the position to the specified point in the multimedia stream.

Run-time access: read/write. Design-time access: not applicable.

Settings

Setting	Description
double	New position within the stream, in seconds.

Data Type

double

CurrentState Property

Control Properties

Describes the playback file's current state.

[*form.*]*object*.**CurrentState**

Remarks

You cannot assign a value to this property. You can change it by calling the Stop, Run, or Pause method, and DirectShow notifies the application of the change by sending the StateChange event.

The **amv** settings are available only to C/C++ programmers; scripters and other programmers should use the numerical values.

Run-time access: read-only. Design-time access: not applicable.

Settings

Setting	Value	Description
amvStopped	0	The player is stopped.
amvPaused	1	The player is paused.
amvRunning	2	The player is playing the multimedia file.

Data Type

Long

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

DisplayBackColor Property

Control Properties

Specifies the display panel's background color.

[*form.*]*object*.**DisplayBackColor** [= *color*]

Remarks

At design time, the default setting is black (0x0).

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte is not 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Object Browser's object library.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
Normal RGB colors	Colors specified by using the color palette or by using the Visual Basic **RGB** or **QBColor** functions in code.
System default colors	Colors specified by system color constants listed in the Object Browser's object library. The Windows® operating environment substitutes the user's choices as specified in the Control Panel settings.

See Also

DisplayForeColor

Previous Home Topic Contents Index Next

DisplayForeColor Property

Control Properties

Specifies the display panel's foreground color.

[*form*.]*object*.**DisplayForeColor** [= *color*]

Remarks

At design time, the default setting is white (&HFFFFFF).

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte is not 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Object Browser's object library.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
Normal RGB colors	Colors specified by using the color palette or by using the Visual Basic **RGB** or **QBColor** functions in code.
System default colors	Colors specified by system color constants listed in the Object Browser's object library. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings.

See Also

DisplayBackColor

Previous Home Topic Contents Index Next

DisplayMode Property

Control Properties

Indicates whether the display panel shows the current position in seconds or frames.

[*form*.]*object*.**DisplayMode** [= *setting*]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
0	(Default) Display the current position in seconds.
1	Display the current position in frames.

Data Type

Integer (Enumerated)

See Also

AllowHideDisplay, ShowDisplay

Previous Home Topic Contents Index Next

EnableContextMenu Property

Control Properties

Indicates whether or not to enable the control's shortcut menu.

[*form*.]*object*.**EnableContextMenu** [= { **True** | **False** }]

Remarks

The shortcut menu appears when the end-user right-clicks anywhere on the control.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
True	(Default) The shortcut menu is enabled.
False	The shortcut menu is disabled.

Data Type

Boolean

See Also

ShowDisplay

Previous Home Topic Contents Index Next

Enabled Property

Control Properties

Specifies whether or not the control is enabled.

[*form.*]*object*.**Enabled** [= *setting*]

Remarks

This property is identical to the standard Visual Basic **Enabled** property.

A disabled control remains visible (if it already was) and will continue to play, if it was already playing. When a control is disabled, none of the commands or menus respond to user input.

Disabling a control at design time is useful if you want to control the media stream from an external source, such as a custom group of controls on a Web page, instead of from the built-in controls panel.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
-1	(Default) The control is enabled.
0	The control is disabled.

Data Type

Integer (Enumerated)

EnablePositionControls Property

Control Properties

Indicates whether or not to show the position buttons in the controls panel.

[*form.*]*object*.**EnablePositionControls** [= { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The position controls are visible.
False	The position controls are not visible.

Data Type

Boolean

See Also

ShowPositionControls

EnableSelectionControls Property

Control Properties

Indicates whether or not to show the selection controls in the controls panel.

[*form*.]*object*.**EnableSelectionControls** [**=** { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The selection controls are visible.
False	The selection controls are not visible.

Data Type

Boolean

See Also

ShowSelectionControls

Previous Home Topic Contents Index Next

EnableTracker Property

Control Properties

Indicates whether or not to show the trackbar control.

[*form*.]*object*.**EnableTracker** [**=** { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The trackbar is visible.
False	The trackbar is not visible.

Data Type

Boolean

See Also

ShowTracker

Previous Home Topic Contents Index Next

FileName Property

Control Properties

Specifies the name of the source data file.

[*form*.]*object*.**FileName** [= *string*]

Remarks

The ActiveMovie Control asynchronously opens the file specified by this property at run time. This means that an application must receive the ReadyStateChange event with an amvInteractive or amvComplete value before attempting to call the Run method.

When designing an application using Microsoft® Visual Basic® that uses the ActiveMovie Control, you can choose the media source file at design time. After you add a control to the application, fill in the full path to the media file as the value for the **FileName** property in Properties. You can also double-click the "..." value in the uninitialized **FileName** property and its property page will appear. If you click the Browse button, you can pick the desired source file from the Open dialog box.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
string	Name of the file that contains the multimedia stream.

Data Type

String

Previous Home Topic Contents Index Next

FilterGraph Property

Control Properties

Contains the IUnknown interface pointer to the current filter graph object.

[*form*.]*object*.**FilterGraph** [= *punk*]

Remarks

The filter graph represents a specific configuration of source, transform, and rendering filters. The filter graph is the complete set of software components needed to process a given multimedia stream within the DirectShow architecture.

You can set this property to change the current filter graph.

The IDispatch interface pointer for the filter graph object is contained in the

FilterGraphDispatch property.

Run-time access: read/write. Design-time access: read-only.

Settings

Setting	Description
punk	IUnknown pointer for the filter graph object.

Data Type

Integer (IUnknown*)

See Also

IFilterGraph Interface in the DirectShow SDK documentation.

FilterGraphDispatch Property

Control Properties

Contains an IDispatch interface pointer to the current filter graph object.

[*form*.]*object*.**FilterGraphDispatch** [= *pdisp*]

Remarks

The filter graph represents a specific configuration of source, transform, and rendering filters. The filter graph represents the complete set of software components needed to process a given multimedia stream within the DirectShow architecture.

Run-time access: read-only. Design-time access: read-only.

Settings

Setting	Description
pdisp	IDispatch pointer to the current filter graph object.

Data Type

Integer (IDispatch*)

See Also

FilterGraph

Previous Home Topic Contents Index Next

FullScreenMode Property

Control Properties

Expands the area of the playback panel to fill the entire screen.

[*form*.]*object*.**FullScreenMode** [**=** { **True** | **False** }]

Remarks

Run-time access: read-only. Design-time access: read/write.

Settings

Setting	Description
True	Use full-screen mode.
False	Do not use full-screen mode.

Data Type

Boolean

See Also

MovieWindowSize

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

MovieWindowSize Property

Control Properties

Specifies the size of the playback panel.

[*form.*]*object*.**MovieWindowSize** [= *setting*]

Remarks

The **amv** settings are available only to C/C++ programmers; scripters and other programmers should use the numerical values.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Value	Description
amvOriginalSize	0	(Default) Uses the authored size.
amvDoubleOriginalSize	1	Increases the image projection size to twice the authored size.
amvOneSixteenthScreen	2	Projects the images onto an area the size of one-sixteenth of the screen.
amvOneFourthScreen	3	Projects the images onto an area the size of one-quarter of the screen.
amvOneHalfScreen	4	Projects the images onto an area the size of half of the screen.

Data Type

Long

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

PlayCount Property

Control Properties

Specifies the number of times to play the multimedia stream.

[*form.*]*object*.**PlayCount** [= *long*]

Remarks

If you set this property to zero, the control will play the multimedia stream repeatedly, restarting as soon as it reaches the end.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
long	Numeric expression that specifies the number of times to play the multimedia stream.

Data Type

Long

See Also

AutoStart

Rate Property

Control Properties

Specifies the playback rate for the multimedia stream.

[*form*.]*object*.**Rate** [= *double*]

Remarks

This property acts as a multiplier value that allows you to play the stream at a faster or slower rate. The value 1.0 indicates the authored speed. Note that the audio track becomes difficult to understand at rates lower than 0.5 and higher than 1.5.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
double	Numeric expression that represents the playback rate; 1.0 corresponds to the authored rate and is the default value.

Data Type

double

ReadyState Property

Control Properties

Specifies the control's state of readiness.

[*form*.]*object*.**ReadyState**

Remarks

The ActiveMovie Control changes this property at run time, and its value is passed as a parameter to the ReadyStateChange event. The run method will only work on the control after the **ReadyState** property has reached the amvInteractive or amvComplete state.

Run-time access: read-only. Design-time access: read-only.

Settings

Setting	Value	Description
amvUninitialized	1	The FileName property has not been initialized.
amvLoading	0	The ActiveMovie Control is asynchronously loading a file.
amvInteractive	3	The control loaded a file, and has downloaded enough data to play the file, but has not yet received all data.
amvComplete	4	All data has been downloaded.

Data Type

Long

SelectionEnd Property

Control Properties

Specifies the ending position in this multimedia stream, in seconds, relative to the stream's beginning.

[*form*.]*object*.**SelectionEnd** [= *double*]

Remarks

The default value for **SelectionEnd** is the length of the multimedia stream; this value is not accessible until the file is loaded completely. When DirectShow sends the ReadyStateChange event with a value of Complete, you can access the value of this property.

Run-time access: read/write. Design-time access: none.

Settings

Setting	Description
double	Numeric expression that specifies the position in the multimedia stream that represents the end of the playback sequence.

Data Type

double

See Also

SelectionStart

SelectionStart Property

Control Properties

Specifies the starting position in this multimedia stream, in seconds, relative to the stream's beginning.

[*form*.]*object*.**SelectionStart** [= *double*]

Remarks

The default value for **SelectionStart** is zero; this value is not accessible until the file is loaded completely. When DirectShow sends the ReadyStateChange event with a value of Complete, you can access the value of this property.

Run-time access: read/write. Design-time access: none.

Settings

Setting	Description
double	Numeric expression that specifies the position within the multimedia stream that represents the beginning of the playback sequence.

Data Type

double

See Also

SelectionEnd

Previous Home Topic Contents Index Next

ShowControls Property

Control Properties

Indicates whether or not the controls panel is visible.

[*form*.]*object*.**ShowControls** [**=** { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
True	(Default) The controls panel is visible.
False	The controls panel is not visible.

Data Type

Boolean

See Also

ShowDisplay

Previous Home Topic Contents Index Next

ShowDisplay Property

Control Properties

Indicates whether or not the display panel is visible.

[*form*.]*object*.**ShowDisplay** [= { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The display panel is visible.
False	The display panel is not visible.

Data Type

Boolean

See Also

ShowControls

Previous Home Topic Contents Index Next

ShowPositionControls Property

Control Properties

Indicates whether or not the position controls are visible.

[*form*.]*object*.**ShowPositionControls** [= { **True** | **False** }]

Remarks

The position controls consist of four buttons: Previous, Rewind, Forward, and Next. Rewind and Forward move the media position rapidly backward and forward. If the current media position is after the position specified in SelectionStart, Previous sets the media position to the value of **SelectionStart**. If not, Previous sets the media position to the start of the data stream. If the current media position is before the position specified in **SelectionStart**, Next sets the media position to the value of **SelectionStart**. If not, Next sets the media position to the end of the data stream.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
True	The position controls are visible.
False	(Default) The position controls are not visible.

Data Type

Boolean

See Also

EnablePositionControls

ShowSelectionControls Property

Control Properties

Indicates whether or not the selection controls are visible.

[*form*.]*object*.**ShowSelectionControls** [**=** { **True** | **False** }]

Remarks

There are two selection buttons—Start Selection and End Selection. Start Selection sets the SelectionStart property to the current media position and End Selection sets the SelectionEnd property to the current media position. If you choose an invalid selection, such as an end time that is earlier than the start time, DirectShow automatically sets **SelectionStart** to the start of the data stream and **SelectionEnd** to the end.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	The selection controls are visible.
False	(Default) The selection controls are not visible.

Data Type

Boolean

See Also

EnableSelectionControls

ShowTracker Property

Control Properties

Indicates whether or not the trackbar is visible.

[*form*.]*object*.**ShowTracker** [= { **True** | **False** }]

Remarks

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	**Description**
True	(Default) The trackbar is visible.
False	The trackbar is not visible.

Data Type

Boolean

See Also

EnableTracker

Volume Property

Control Properties

Specifies the volume, in hundredths of decibels.

[*form*.]*object*.**Volume** [= *long*]

Remarks

The value ranges from –10,000 to 0. The value 0 is full volume and –10,000 is no volume. Not all devices support 10,000 discrete steps of volume.

Run-time access: read/write. Design-time access: read/write.

Settings

Setting	Description
long	Numeric expression that specifies the audio volume, in hundredths of decibels. The default value is zero.

Data Type

Long

See Also

Balance Property

Control Methods

The ActiveMovie Control supports the following methods.

Method	Description
AboutBox	Displays version and copyright information about the ActiveMovie Control.
IsSoundCardEnabled	Determines whether the computer's sound card is enabled.
Pause	Suspends a play operation without changing the current position.
Run	Starts a multimedia stream from the specified starting position or continues playing a paused stream.
Stop	Stops playback and resets the position as indicated by the AutoRewind and SelectionStart properties.

The ActiveMovie Control also supports several methods that are common to other controls: **Drag**, **Move**, **SetFocus**, **ShowWhatsThis**, and **ZOrder**. For information about these methods, see documentation for Visual Basic.

Previous Home Topic Contents Index Next

AboutBox Method

Control Methods

Displays version and copyright information about the ActiveMovie Control.

object.AboutBox

Previous Home Topic Contents Index Next

IsSoundCardEnabled Method

Control Methods

Determines whether the computer's sound card is enabled.

object.IsSoundCardEnabled [= { **True** | **False** }]

Remarks

The **IsSoundCardEnabled** method determines whether the computer has one or more sound cards enabled.

Run-time access: read. Design-time access: read.

Settings

Setting	**Description**
True	Sound card is enabled.
False	No sound cards are enabled.

Data Type

Boolean

Previous Home Topic Contents Index Next

Pause Method

Control Methods

Suspends a play operation without changing the current position.

object.Pause

Remarks

The **Pause** method pauses the multimedia stream at the current position. To continue playing the multimedia stream, use the Run method.

To stop the multimedia stream, use the Stop method.

Previous Home Topic Contents Index Next

Run Method

Control Methods

Starts a multimedia stream from the specified starting position or continues playing a paused stream.

object.Run

Remarks

The **Run** method starts the multimedia stream at the starting position specified by the SelectionStart property. In the absence of other user or application input, the **Run** method continues playing the stream to the position specified by the SelectionEnd property.

The **Run** method is also used to resume playing a paused multimedia stream.

To pause playing, call the Pause method. To stop playing, call the Stop method.

Previous Home Topic Contents Index Next

Stop Method

Control Methods

Stops playback and resets the position as indicated by the AutoRewind and SelectionStart properties.

object.Stop

Remarks

The **Stop** method changes the CurrentState property.

When the **Stop** method halts a play operation, you can reset the current position to the starting point for the multimedia stream, as indicated by the SelectionStart property. The AutoRewind property determines whether the position is reset to this starting position.

To halt a play operation without changing the current position, use the Pause method.

See Also

Run Method

Previous Home Topic Contents Index Next

Control Events

The ActiveMovie Control supports the following events.

Event	Description
DisplayModeChange	Indicates changes to the DisplayMode property.
Error	Indicates that an error occurred.
OpenComplete	Specifies the state of readiness for this ActiveMovie Control, based on how completely the source file has loaded.
PositionChange	Indicates changes to the current media position, such as when the user selects (or seeks to) a new media position.
ReadyStateChange	Indicates changes to the control's state of readiness.
StateChange	Indicates player state changes.
Timer	Handles timer events.

The ActiveMovie Control also supports several events common to other controls: **Click**, **DblClick**, **KeyDown**, **KeyPress**, **KeyUp**, **MouseDown**, **MouseMove**, and **MouseUp**. For information about these events, see the event documentation for Visual Basic.

Previous Home Topic Contents Index Next

DisplayModeChange Event

Control Events

Indicates changes to the DisplayMode property, such as from full-screen to half.

Private Sub object_DisplayModeChange()

Remarks

The **DisplayModeChange** event is raised when the DisplayMode property changes. The current display mode appears in the **DisplayMode** property. State flags are listed under **DisplayMode** property.

Previous Home Topic Contents Index Next

Error Event

Control Events

Indicates that an error occurred.

Private Sub object_Error(ByVal SCode As Integer, ByVal Description As String, ByVal Source As String, CancelDisplay As Boolean)

Parameters

object
  Object expression that evaluates to an ActiveMovie Control.
SCode
  Error code.
Description
  String describing the error that occurred.
Source
  String containing the ActiveMovie Control's name.
CancelDisplay
  Value that the client can set to cancel the default error messages.

Remarks

The **Error** event is sent when DirectShow reports an error during playback. By default, the ActiveMovie Control displays a message box containing the description string. To avoid displaying this box, set the *CancelDisplay* parameter of the **Error** event to False.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

OpenComplete Event

Control Events

Indicates the control finished opening the media file and is ready for playback.

Private Sub object_OpenComplete()

Parameters

object
: Object expression that evaluates to an ActiveMovie Control.

Remarks

This event is provided for backward compatibility; use the ReadyStateChange event instead.

Previous Home Topic Contents Index Next

PositionChange Event

Control Events

Indicates changes to the current media position, such as when the user selects (or seeks to) a new media position.

Private Sub object_PositionChange(ByVal oldPosition As Double, ByVal newPosition As Double)

Parameters

object
: Object expression that evaluates to an ActiveMovie Control.

oldPosition
: Position before it changed, in seconds.

newPosition
: Current position, in seconds, after the position change occurred.

Remarks

Changes made directly to the CurrentPosition property do not trigger this event.

Previous Home Topic Contents Index Next

ReadyStateChange Event

Control Events

Indicates changes to the control's state of readiness.

Private Sub object_ReadyStateChange(ByVal ReadyState As Long)

Parameters

object
Object expression that evaluates to an ActiveMovie Control.
ReadyState
Current state of readiness, after the change occurred.

Remarks

When you change the FileName property, applications should check the control's state of readiness before attempting to call the Run method on the newly opened file. Applications should call the **Run** method only if the *ReadyState* parameter equals amvInteractive or amvComplete.

Settings

Following are possible settings for the *ReadyState* parameter.

Setting	Value	Description
amvUninitialized	1	The FileName property has not been initialized.
amvLoading	0	The ActiveMovie Control is asynchronously loading a file.
amvInteractive	3	The control loaded a file, and downloaded enough data to play the file, but has not yet received all data.
amvComplete	4	All data has been downloaded.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

StateChange Event

Control Events

Indicates that the player state changed, such as from running to stopped.

Private Sub object_StateChange(ByVal oldState As Long, ByVal newState As Long)

Parameters

object

Object expression that evaluates to an ActiveMovie Control.

oldState

Previous state, before the change occurred.

newState

Current state, after the change occurred.

Remarks

The ActiveMovie Control raises the **StateChange** event when the player state changes, such as from stopped to running. The current player state appears in the CurrentState property. You can find the player state flags in the **CurrentState** property documentation.

Settings

Following are possible settings for the *oldState* or *newState* parameters.

Setting	Value	Description
amvStopped	0	The player is stopped.
amvPaused	1	The player is paused.
amvRunning	2	The player is playing the multimedia file.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Timer Event

Control Events

Provides the rate at which DirectShow refreshes the control panel's display.

Private Sub object_Timer( )

Parameters

object

Object expression that evaluates to an ActiveMovie Control.

Remarks

The **Timer** event is raised at the intervals specified by the control's timer.

Previous Home Topic Contents Index Next

Control Property Sheet

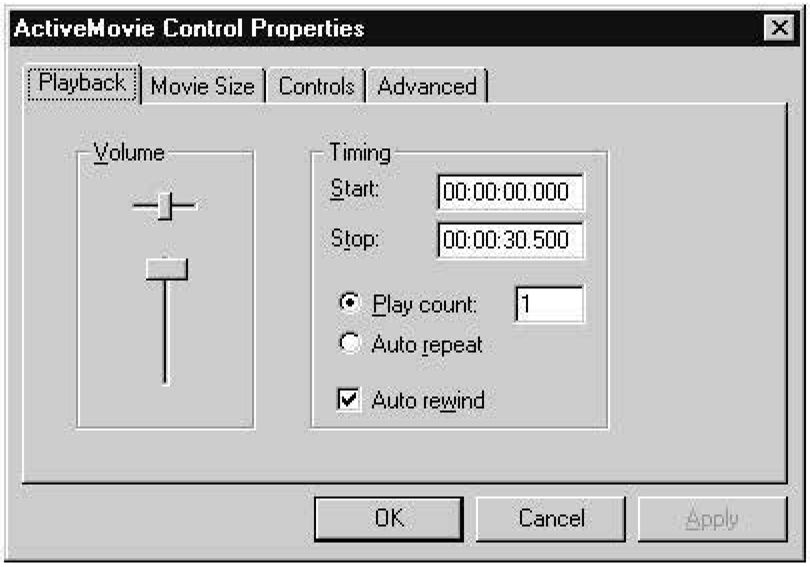
A property sheet is a dialog box with tabbed pages that you can use to set properties on a control. You can access the ActiveMovie Control Properties property sheet at either design or run time.

To activate a property page at design time, select the control, press F4, then click Custom in the Properties page. To activate a property page at run time, right-click, and then click Properties on the shortcut menu that appears. The following is a list of the tabs in the ActiveMovie Control Properties property sheet.

- Playback
- Movie Size
- Controls
- Advanced

Playback Tab

Use the controls on the Playback tab at run time to set options such as volume, rate of play, and the number of times you want the current video to play.



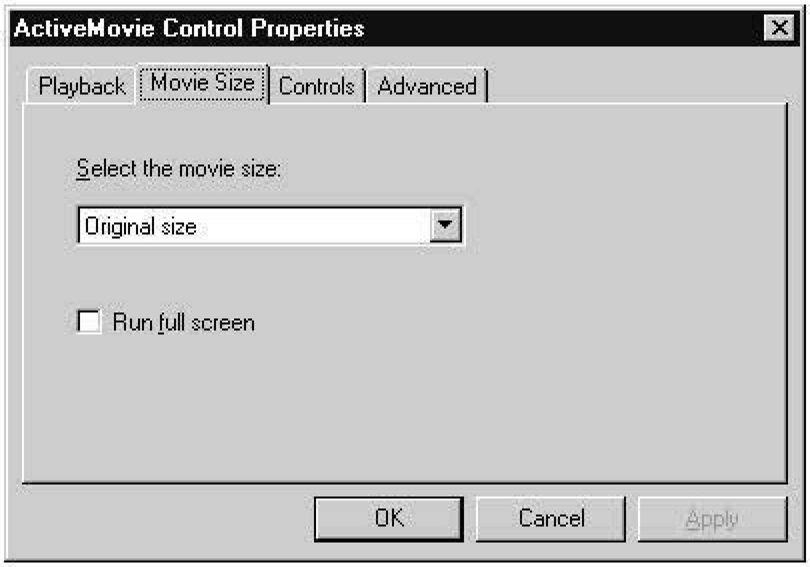
Timing
Start: 00:00:00.000
Stop: 00:00:30.500
[✓ Play count:] 1
Auto repeat
[✓ Auto rewind]

Property	Description
Volume	Adjusts the control's volume. For information about how to set this property in code, see Volume.
Balance	Adjusts balance for the control.
Start Time	Identifies the start time for the current file.
Stop Time	Identifies the stop time for the current file. For information about how to set this property in code, see the AllowChangeDisplayMode property.
Play Count	Specifies the number of times to play the current file consecutively. For information about how to set this property in code, see the PlayCount property.

Auto Repeat	Causes the ActiveMovie Control to play the current file repeatedly. For information about how to set this property in code, see the AutoStart property.
Auto Rewind	Indicates whether to rewind the multimedia stream automatically and reposition at the beginning after playing stops. For information about how to set this property in code, see the AutoRewind property.

Movie Size Tab

Use the controls on the Movie Size tab at run time to specify the size of the window in which you want to play video.


ActiveMovie Control Properties

Playback | Movie Size | Controls | Advanced

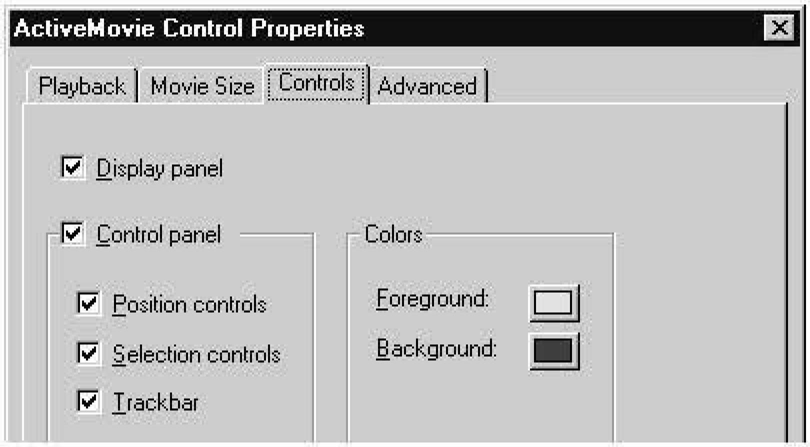
Select the movie size: Original size

☐ Run full screen

Property	Description
Movie Size	Use this option list to specify the movie's size. For information about how to set this property in code, see the MovieWindowSize property.
Run Full Screen	Select this option to maximize the ActiveMovie Control so that it fills the entire screen. For information about how to set this property in code, see the FullScreenMode property.

Controls Tab

Use the controls on the Controls tab at run time to specify which ActiveMovie controls you want to be visible.


ActiveMovie Control Properties

Playback | Movie Size | Controls | Advanced

☑ Display panel

☑ Control panel

☑ Position controls

☑ Selection controls

☑ Trackbar

Colors

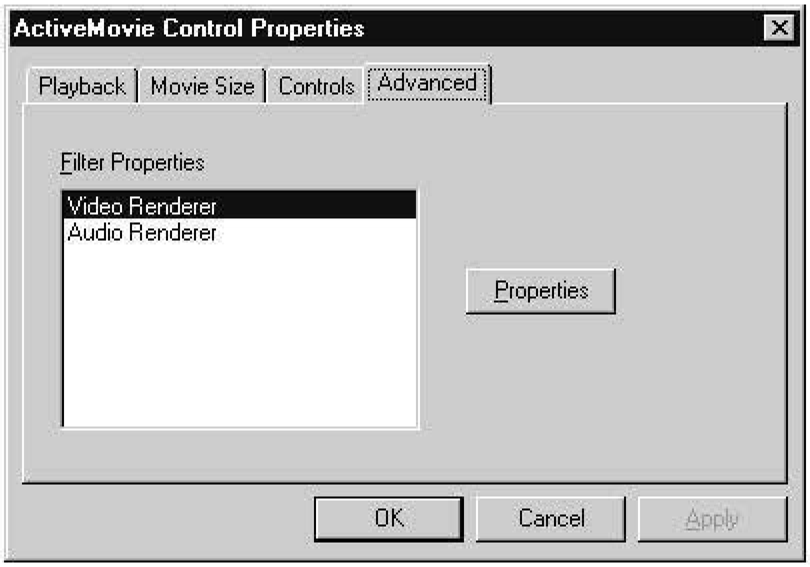
Foreground:

Background:

Property	**Description**
Display Panel	Select this check box if you want the display panel to be visible at run time. For information about how to set this property in code, see the ShowDisplay property.
Control Panel	Select this check box if you want the control panel to be visible at run time. For more information about how to set this property in code, see the ShowControls property.
Position Controls	Select this check box if you want the video stream position to be displayed at run time. For information about how to set this property in code, see the DisplayMode property.
Selection Controls	Select this check box to make the selection controls visible. For information about how to set this property in code, see the SelectionStart property.
Show Trackbar	Select this check box to toggle the visibility of the trackbar. For information about how to set this property in code, see the ShowTracker property.
Foreground Color	Use this control to show and set the foreground color of the ActiveMovie Control. For information about how to set this property in code, see the DisplayForeColor property.
Background Color	Use this control to show and set the background color of the ActiveMovie Control. For information about how to set this property in code, see the DisplayBackColor property.

Advanced Tab

Use the Advanced tab to run a filter on the currently playing video.



Property	**Description**
Filter Properties	Selects the filter property page to display when the Properties button is clicked.
Properties	Displays the property page of the filter selected in the Filter Properties list. Property pages allow you to change parameters on individual filters in the DirectShow filter graph. For more information about properties of DirectShow filters, see Filters in the DirectShow SDK documentation.

Previous Home Topic Contents Index Next

Control Shortcut Keys

You can use the following key combinations to activate ActiveMovie Control commands.

Key(s)	**Result**
CTRL+D	Toggle display
CTRL+LEFT ARROW	Rewind
CTRL+P	Pause
CTRL+R	Run
CTRL+RIGHT ARROW	Forward
CTRL+S	Stop
CTRL+SHIFT+LEFT ARROW	Previous
CTRL+SHIFT+RIGHT ARROW	Next
CTRL+T	Toggle control panel
ALT+ENTER	Toggle between full-screen and windowed modes
SHIFT+window resize	Stretches video to fill the control window

Previous Home Topic Contents Index Next

Visual Basic Objects

DirectShow interfaces on the filter graph manager are available to applications based on Microsoft® Visual Basic® version 5.*x*. A Visual Basic-based application can use these interfaces (called objects in Visual Basic) to build and control DirectShow filter graphs.

IAMCollection Object

The filter graph manager exposes the **IAMCollection** object, which allows access to object collections. These collections contain IPinInfo, IFilterInfo, or IMediaTypeInfo objects.

Properties

Name	Description
Count	Returns the number of items in the collection.

Methods

Name	Description
Item	Retrieves the specified member of the collection.

Count Property (IAMCollection Object)

IAMCollection Object

Returns the number of items in the collection.

objCollection.**Count**

Parts

objCollection
    Object expression that evaluates to an IAMCollection object.

Previous Home Topic Contents Index Next

Item Method (IAMCollection Object)

IAMCollection Object

Retrieves the specified collection member and stores it in the passed-in object.

objCollection.**Item** *lItem*, *objNew*

Parts

objCollection
    Object expression that evaluates to an IAMCollection object.
lItem
    Index of the item to retrieve.
objNew
    Object expression that can evaluate to an IFilterInfo, IPinInfo, or IMediaTypeInfo object depending on the collection type. This argument will contain the retrieved item.

Remarks

The index value ranges from 0 to (Count – 1).

Previous Home Topic Contents Index Next

IBasicAudio Object

The **IBasicAudio** object supports the audio component of the filter graph. It allows access to volume and balance functionality.

The Volume property is a value between -10,000 and 0 representing a set of linear steps. Not all devices support 10,000 distinguishable steps.

The Balance property is a value between -10,000 and 10,000. A value of -10,000 indicates that the right speaker has been disabled and only the left speaker is receiving an audio signal. A value of 0 indicates that both speakers are receiving equivalent audio signals. A value of 10,000 indicates that the left speaker has been disabled and only the right speaker is receiving an audio signal.

Properties

Name	Description
Balance	Sets or returns the balance for the audio signal.
Volume	Sets or returns the volume (amplitude) of the audio signal.

Balance Property (IBasicAudio Object)

IBasicAudio Object

Sets or returns the balance of the audio channel.

objAudio.**Balance** [= *lValue*]

Parts

objAudio
    Object expression that evaluates to an IBasicAudio object.
lValue
    New value for the **Balance** property.

Remarks

The **Balance** property is a value between -10,000 and 10,000. A value of -10,000 indicates that the right speaker has been disabled and only the left speaker is receiving an audio signal. A value of 0 indicates that both speakers are receiving equivalent audio signals. A value of

10,000 indicates that the left speaker has been disabled and only the right speaker is receiving an audio signal.

Note that not all devices support 10,000 distinguishable steps.

If you try to set the **Balance** property to values outside this range, the IBasicAudio object raises run-time error 5.

Previous Home Topic Contents Index Next

Volume Property (IBasicAudio Object)

IBasicAudio Object

Sets or returns the volume of the audio channel.

objAudio.**Volume** [= *lValue*]

Parts

objAudio
Object expression that evaluates to an IBasicAudio object.
lValue
New value for the **Volume** property.

Remarks

The allowable input range is –10,000 to 0.

Full volume is 0; –10,000 is silence. The scale is linear. Values outside this range will return run-time error 5.

Note that not all devices support 10,000 distinguishable steps.

Previous Home Topic Contents Index Next

IBasicVideo Object

The filter graph manager exposes the **IBasicVideo** object, which supports the video properties of a generic video window. Generally, this is a video renderer that draws video into a window on the display.

The video renderer must be connected to use **IBasicVideo** object methods. If it isn't connected, all methods return run-time error 521. Properties set on a video renderer generally persist between successive connections and disconnections. All applications should ensure that they reset the renderer properties before starting a presentation.

When working with video, the application can select a portion of the video to use. This portion is the source rectangle that the **IBasicVideo** object controls. **IBasicVideo** allows the source rectangle to be set and retrieved. All **IBasicVideo** rectangles have top, left, width, and height properties. When no source rectangle has been set, the properties of the source rectangle return the full, native video size.

Properties

Name	**Description**
AvgTimePerFrame	Retrieves the average time between successive frames in 100-nanosecond units.
BitErrorRate	Retrieves an approximate bit error rate for the video stream.
BitRate	Retrieves an approximate bit rate for the video stream.
DestinationLeft	Sets or retrieves the x-axis coordinate for the destination video rectangle.
DestinationHeight	Sets or retrieves the height of the destination video rectangle.
DestinationTop	Sets or retrieves the y-axis coordinate for the destination video rectangle.
DestinationWidth	Sets or retrieves the width of the destination video rectangle.
SourceHeight	Sets or retrieves the height of the source video rectangle.
SourceLeft	Sets or retrieves the x-axis coordinate for the source video rectangle.
SourceTop	Sets or retrieves the y-axis coordinate for the source video rectangle.
SourceWidth	Sets or retrieves the width of the source video rectangle.
VideoHeight	Retrieves the current video height.
VideoWidth	Retrieves the current video width.

Methods

Name	**Description**
GetCurrentImage	Returns a copy of the image that is currently waiting at the renderer.
GetDestinationPosition	Retrieves the destination rectangle for the window.
GetSourcePosition	Retrieves the source video rectangle.
GetVideoPaletteEntries	Retrieves the color palette entries required by the video.
GetVideoSize	Retrieves the native video dimensions.
IsUsingDefaultSource	Indicates whether the default source is being used.
SetDefaultDestinationPosition	Sets the default destination position for the window.
SetDefaultSourcePosition	Informs the renderer to use the default source rectangle.
SetDestinationPosition	Sets the destination rectangle for the window.
SetSourcePosition	Sets the source video rectangle.

Previous Home Topic Contents Index Next

AvgTimePerFrame Property (IBasicVideo Object)

IBasicVideo Object

Retrieves the average time between successive frames in 100-nanosecond units.

objVideo.**AvgTimePerFrame**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

BitErrorRate Property (IBasicVideo Object)

IBasicVideo Object

Retrieves a bit error rate for the video stream (one error for approximately this many bits).

objVideo.**BitErrorRate**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

BitRate Property (IBasicVideo Object)

IBasicVideo Object

Retrieves an approximate bit rate for the video stream (in bits per second).

objVideo.**BitRate**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

Previous Home Topic Contents Index Next

DestinationHeight Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the height of the destination rectangle.

objVideo.**DestinationHeight** [= *lValue*]

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.
lValue
New value for the destination rectangle height.

Remarks

Setting this coordinate does not affect the destination rectangle y-axis origin.

Previous Home Topic Contents Index Next

DestinationLeft Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the destination x-axis origin coordinate.

objVideo.**DestinationLeft** [= *lValue*]

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
lValue
    New value of the destination x-axis origin.

Remarks

Setting this coordinate does not affect the destination rectangle width.

Previous Home Topic Contents Index Next

DestinationTop Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the destination y-axis origin coordinate.

objVideo.**DestinationTop** [= *lValue*]

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
lValue
    New value of the y-axis origin.

Remarks

Setting this coordinate does not affect the destination rectangle height.

Previous Home Topic Contents Index Next

DestinationWidth Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the width of the destination rectangle.

objVideo.**DestinationWidth** [= *lValue*]

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
lValue
    New width value.

Remarks

Setting this coordinate does not affect the destination rectangle x-axis origin.

Previous Home Topic Contents Index Next

GetCurrentImage Method (IBasicVideo Object)

IBasicVideo Object

Retrieves the image currently waiting at the renderer.

objVideo.**GetCurrentImage** *BufferSize*, *DIBImage*

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
BufferSize
    Long value that specifies the size of the buffer the caller is passing in. **GetCurrentImage** sets this value to the actual number of bytes in the buffer before exiting.
DIBImage
    Long value that will contain the buffer where the complete image will be stored in device-independent bitmap (DIB) format. You must pass the buffer by reference.

Remarks

Pause the video renderer before calling this method; calling this method in any other state returns run-time error 549. Depending on what data the source filter has available, the video renderer is not guaranteed to service this request.

An application can use this method to get a copy of the current image the video renderer holds when paused by calling the method with *DIBImage* set to Nothing.

Previous Home Topic Contents Index Next

GetDestinationPosition Method (IBasicVideo Object)

IBasicVideo Object

Retrieves the position of the destination rectangle in window coordinates.

objVideo.**GetDestinationPosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
Left
    Long integer value that will contain the destination window's x-axis origin.
Top
    Long integer value that will contain the destination window's y-axis origin.
Width
    Long integer value that will contain the destination window's width.
Height
    Long integer value that will contain the destination window's height.

Remarks

This method has the same effect as individually retrieving the DestinationLeft, DestinationTop, DestinationWidth, and DestinationHeight properties.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

GetSourcePosition Method (IBasicVideo Object)

IBasicVideo Object

Retrieves the source position that is clipped to the available video.

objVideo.**GetSourcePosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
Left
    Long integer value that will contain the source window's x-axis origin.
Top
    Long integer value that will contain the source window's y-axis origin.
Width
    Long integer value that will contain the source window's width.
Height
    Long integer value that will contain the source window's height.

Remarks

This method has the same effect as retrieving the values of the SourceLeft, SourceTop, SourceWidth, and SourceHeight properties.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

GetVideoPaletteEntries Method (IBasicVideo Object)

IBasicVideo Object

Retrieves the palette colors for the video.

objVideo.**GetVideoPaletteEntries** *StartIndex*, *Entries*, *Retrieved*, *Palette*

Parts

objVideo

Object expression that evaluates to an IBasicVideo object.

StartIndex
: Long integer value that specifies the start index for the palette.

Entries
: Long integer value that specifies the number of palette entries required.

Retrieved
: Long integer value that will contain the number of entries actually returned in *Palette*.

Palette
: Long integer value that will contain a pointer to an array of Microsoft® Win32® PALETTEENTRY structures.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

GetVideoSize Method (IBasicVideo Object)

IBasicVideo Object

Retrieves the native video dimensions.

objVideo.**GetVideoSize** *Width***,** *Height*

Parts

objVideo
: Object expression that evaluates to an IBasicVideo object.

Width
: Long integer value that will contain the video window's width.

Height
: Long integer value that will contain the video window's height.

Remarks

Retrieves the native video width and height, disregarding any source rectangle that might have been set. ActiveX™ Controls or other applications can use this information to negotiate space in compound documents.

Previous Home Topic Contents Index Next

IsUsingDefaultSource Method (IBasicVideo Object)

IBasicVideo Object

Retrieves whether or not this object is using the default source.

objVideo.**IsUsingDefaultSource**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

Return Values

True indicates that the IBasicVideo object is using the default source; False indicates that it is not.

SetDefaultDestinationPosition Method (IBasicVideo Object)

IBasicVideo Object

Sets the default destination position of the video window so that the renderer uses the entire window for playback.

objVideo.**SetDefaultDestinationPosition**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

SetDefaultSourcePosition Method (IBasicVideo Object)

IBasicVideo Object

Informs the renderer to use the default source rectangle.

objVideo.**SetDefaultSourcePosition**

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.

Previous Home Topic Contents Index Next

SetDestinationPosition Method (IBasicVideo Object)

IBasicVideo Object

Sets the destination rectangle in window coordinates.

objVideo.**SetDestinationPosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
Left
    Long value specifying the x-axis origin of the destination window, in pixels.
Top
    Long value specifying the y-axis origin of the destination window, in pixels.
Width
    Long value specifying the width of the destination window, in pixels.
Height
    Long value specifying the height of the destination window, in pixels.

Remarks

This method has the same effect as setting the DestinationLeft, DestinationTop, DestinationWidth, and DestinationHeight properties individually.

SetSourcePosition Method (IBasicVideo Object)

IBasicVideo Object

Sets the source rectangle and is clipped against the available video.

objVideo.**SetSourcePosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideo
: Object expression that evaluates to an IBasicVideo object.

Left
: Long value specifying the x-axis origin of the source window.

Top
: Long value specifying the y-axis origin of the source window.

Width
: Long value specifying the width of the source window.

Height
: Long value specifying the height of the source window.

Remarks

This method has the same effect as setting the SourceLeft, SourceTop, SourceWidth, and SourceHeight properties individually.

SourceHeight Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the height of the source rectangle.

objVideo.**SourceHeight** [= *lValue*]

Parts

objVideo
: Object expression that evaluates to an IBasicVideo object.

lValue
: Long value specifying the current rectangle height, in pixels.

Remarks

Setting the height of the source rectangle has no effect on the width and other source values.

SourceLeft Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the source rectangle's x-coordinate.

objVideo.**SourceLeft** [= *lValue*]

Parts

objVideo
: Object expression that evaluates to an IBasicVideo object.

lValue
: New value for the x-axis origin of the source rectangle.

Remarks

Changes to this property do not affect the width and other source values.

SourceTop Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the source rectangle y-axis origin coordinate.

objVideo.**SourceTop** [= *lValue*]

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
lValue
    Long value specifying the distance from the top of the display to the top of the current window, in pixels.

Remarks

Setting this method has no effect on the source rectangle height.

Previous Home Topic Contents Index Next

SourceWidth Property (IBasicVideo Object)

IBasicVideo Object

Retrieves or sets the source rectangle width.

objVideo.**SourceWidth** [= *lValue*]

Parts

objVideo
    Object expression that evaluates to an IBasicVideo object.
lValue
    Long value specifying the current width, in pixels.

Remarks

This method has no effect on the source rectangle x-axis coordinate.

Previous Home Topic Contents Index Next

VideoHeight Property (IBasicVideo Object)

IBasicVideo Object

Retrieves the native height of the video.

objVideo.**VideoHeight**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

Remarks

This method returns the actual height of the video supplied to the renderer. It does not address any set source rectangle, but simply returns the native vertical dimension. It can be used by applications to negotiate size requirements with in-place container documents.

Previous Home Topic Contents Index Next

VideoWidth Property (IBasicVideo Object)

IBasicVideo Object

Retrieves the native width of the video.

objVideo.**VideoWidth**

Parts

objVideo
Object expression that evaluates to an IBasicVideo object.

Remarks

This method returns the actual width of the video supplied to the renderer. It does not address any set source rectangle, but simply returns the native horizontal dimension. It can be used by applications to negotiate size requirements with in-place container documents.

IFilterInfo Object

IFilterInfo is an object that manages information about a filter and allows access to the filter object and IPinInfo object.

Properties

Name	**Description**
Filename	Retrieves or sets the file name associated with the source filter.
Filter	Retrieves the actual filter object.
IsFileSource	Indicates whether the filter is a file source filter.
Name	Retrieves the filter name.
Pins	Retrieves an IAMCollection object containing the IPinInfo objects for this filter.
VendorInfo	Retrieves a string containing optional information supplied by a vendor about the specified filter.

Methods

Name	**Description**
FindPin	Locates a pin, given an identifier, and returns an object supporting IPinInfo.

Filename Property (IFilterInfo Object)

IFilterInfo Object

Retrieves or sets the name of the file containing the filter.

objFilterInfo.**Filename** [= *string*]

Parts

objFilterInfo
    Object expression that evaluates to an IFilterInfo object.
string

String expression that specifies the new file name.

Previous Home Topic Contents Index Next

Filter Property (IFilterInfo Object)

IFilterInfo Object

Retrieves the filter object.

Set *objFilter* = *objFilterInfo*.**Filter**

Parts

objFilter
Object expression that evaluates to an IBaseFilter object.
objFilterInfo
Object expression that evaluates to an IFilterInfo object.

Remarks

No programmable object interface is offered for the IBaseFilter interface. However, Visual Basic® programmers can use the returned object as a parameter in calls to functions that take an **IBaseFilter** interface pointer as a parameter.

Previous Home Topic Contents Index Next

FindPin Method (IFilterInfo Object)

IFilterInfo Object

Locates a pin, given an identifier, and returns an IPinInfo object.

objFilterInfo.**FindPin** *string*, *objPin*

Parts

objFilterInfo
Object expression that evaluates to an IFilterInfo object.

string
String that contains the name of the pin to find.
objPin
IPinInfo object that will contain the specified pin, if the pin exists.

Remarks

No programmable object interface is offered for the IPin interface. However, Visual Basic programmers can use the returned object as a parameter in calls to functions that take the **IPin** interface pointer as a parameter.

Previous Home Topic Contents Index Next

IsFileSource Property (IFilterInfo Object)

IFilterInfo Object

Determines if the file contains the source video.

objFilterInfo.**IsFileSource**

Parts

objFilterInfo
Object expression that evaluates to an IFilterInfo object.

Return Values

The value True indicates that the file contains the source video. False indicates that the file does not contain the source video.

Previous Home Topic Contents Index Next

Name Property (IFilterInfo Object)

IFilterInfo Object

Retrieves the filter name.

objFilterInfo.**Name**

Parts

objFilterInfo
Object expression that evaluates to an IFilterInfo object.

Remarks

The vendor specifies the filter name.

Previous Home Topic Contents Index Next

Pins Property (IFilterInfo Object)

IFilterInfo Object

Retrieves an IAMCollection object containing the IPinInfo objects for this filter.

Set *objPinCollection* = *objFilterInfo*.**Pins**

Parts

objPinCollection
IAMCollection object that will contain the specific filter's pins.
objFilterInfo
IFilterInfo object that specifies the filter whose pins you want to retrieve.

Remarks

You can use the Count property and Item method to iterate through each of the pin objects in the collection.

Previous Home Topic Contents Index Next

VendorInfo Property (IFilterInfo Object)

IFilterInfo Object

Retrieves a string containing optional information supplied by a vendor about the specified filter.

objFilterInfo.**VendorInfo**

Parts

objFilterInfo
    Object expression that evaluates to an IFilterInfo object.

Remarks

This property contains any information the vendor chooses to supply. It is usually descriptive information about the filter.

Previous Home Topic Contents Index Next

IMediaControl Object

The filter graph exposes the IMediaControl object to allow applications to control the streaming of media through the filters in the graph. The interface provides methods for running, pausing, and stopping the streaming of data. It also provides applications with a simple method that builds graphs to play back media files.

Properties

Name	**Description**
FilterCollection	Retrieves a collection of IFilterInfo objects representing the current filters in the graph.
RegFilterCollection	Retrieves a collection of IRegFilterInfo objects representing the filters available in the registry.

Methods

Name	**Description**
AddSourceFilter	Adds to the graph the source filter that can read the given file name.
GetState	Retrieves the state of the filter graph.
Pause	Pauses all filters in the filter graph.
RenderFile	Adds and connects all filters needed to play the specified file.
Run	Switches the entire filter graph into run mode.
Stop	Switches all filters in the filter graph to a stopped state.

Previous Home Topic Contents Index Next

AddSourceFilter Method (IMediaControl Object)

IMediaControl Object

Adds the source filter that can read the given file name to the graph and returns the created IFilterInfo object.

objMediaControl.**AddSourceFilter** *string*, *objFilterInfo*

Parts

objMediaControl
   Object expression that evaluates to an IMediaControl object.
string
   Name of the file containing the source media.
objFilterInfo
   IFilterInfo object; the source filter created for the specified source file.

FilterCollection Property (IMediaControl Object)

IMediaControl Object

Retrieves a collection of IFilterInfo objects representing the filters in the graph. Returns an IAMCollection object.

Set *objCollection* = *objMediaControl*.**FilterCollection**

Parts

objCollection
   IAMCollection object that represents the collection of filters currently present in the filter graph.
objMediaControl
   Object expression that evaluates to an IMediaControl object.

GetState Method (IMediaControl Object)

IMediaControl Object

Returns a Long integer indicating the current state value.

objMediaControl.**GetState** *msTimeout***,** *State*

Parts

objMediaControl
Object expression that evaluates to an IMediaControl object.
msTimeout
Duration of the time-out, in milliseconds.
State
Long value that will contain the current state. This argument will be one of the following:

Value	**Description**
State_Paused	The media source is paused.
State_Running	The media source is running.
State_Stopped	The media source is stopped.

Return Values

Returns run-time error 567 if the state transition is not complete, or 0 if it completed successfully.

Remarks

Not all state transitions are synchronous. For example, even though the Pause method returns immediately, the graph typically does not complete the transition into paused mode until data is ready at the renderer. This method will not return zero until the state transition has been completed.

If you specify a nonzero time-out, the method waits up to that number of milliseconds for the graph to leave the intermediate state. If the time-out expires before the state transition is complete, the return code will be 567, and the returned state will be the state into which the graph is transitioning (either State_Stopped, State_Paused, or State_Running).

RegFilterCollection Property (IMediaControl Object)

IMediaControl Object

Retrieves a collection of IRegFilterInfo objects representing the filters available in the registry.

Set *objCollection* = *objMediaControl*.**RegFilterCollection**

Parts

objCollection
  IAMCollection object that contains IRegFilterInfo objects.
objMediaControl
  Object expression that evaluates to an IMediaControl object.

Return Values

Returns an IAMCollection object.

Previous Home Topic Contents Index Next

Pause Method (IMediaControl Object)

IMediaControl Object

Pauses all the filters in the filter graph.

objMediaControl.**Pause**

Parts

objMediaControl
  Object expression that evaluates to an IMediaControl object.

Remarks

In the paused state, filters process data but do not render it. Data is pushed down the filter graph and is processed by transform filters as far as buffering permits. No data is rendered (except that media types capable of being rendered statically, such as video, have a static, poster frame rendered in paused mode). Therefore, putting a filter graph into a paused state cues the graph for immediate rendering when put into a running state.

RenderFile Method (IMediaControl Object)

IMediaControl Object

Adds and connects filters needed to play the specified file.

objMediaControl.**RenderFile** *string*

Parts

objMediaControl
Object expression that evaluates to an IMediaControl object.
string
Name of the file to render.

Remarks

This method allows an application to pass the name of a media file that it wants rendered to the filter graph manager. The filter graph manager will build a graph of the filters that are needed to play back this file.

Run Method (IMediaControl Object)

IMediaControl Object

Switches the entire filter graph into a running state.

objMediaControl.**Run**

Parts

objMediaControl

Object expression that evaluates to an IMediaControl object.

Remarks

If the filter graph is in the stopped state, this method first pauses the graph before running.

If an error value is returned, some filters within the graph might have successfully entered the running state. In a multistream graph, entire streams might be playing successfully. The application must determine whether to stop execution or not.

Previous Home Topic Contents Index Next

Stop Method (IMediaControl Object)

IMediaControl Object

Switches all filters in the filter graph to a stopped state.

objMediaControl.**Stop**

Parts

objMediaControl
Object expression that evaluates to an IMediaControl object.

Remarks

In this mode, filters release resources and no data is processed. If the filters are in a running state, this method pauses them before stopping them. This allows video renderers to make a copy of the current frame for poster frame display while stopped.

Previous Home Topic Contents Index Next

IMediaEvent Object

This object supports event notification from the filter graph and the filters within it to the application.

An event code and two parameters represent event notification information. This can be used for typical completion of asynchronous operations, errors that occur during asynchronous operation, or user-initiated events, such as when a user clicks a hot spot.

Filters within the filter graph and the filter graph itself raise event notifications. Possible events include playback completion or asynchronous playback errors. In addition, the filter graph provides a method to generate events at specific reference clock times.

Event notifications are placed in a queue. An application calls the **IMediaEvent** object's GetEvent method to retrieve the next notification from the queue. This method blocks until there is an event to return. The **GetEvent** method's time-out parameter allows the application to specify the time, in milliseconds, to wait for an event, including values of zero (do not wait) and infinite (wait forever).

In addition, applications can retrieve the event handle. GetEventHandle returns a handle to a manual-reset event created by the Microsoft® Win32® CreateEvent function. This event is in a signaled state as long as there are event notifications to collect. The event is cleared by the GetEvent method when there are no more event notifications to collect. This allows an application to use an application programming interface (API), such as MsgWaitForMultipleObjects, to wait for events and other occurrences at the same time. This event handle will be closed when the filter graph is released; therefore, applications should ensure that they are not using it after this point.

The filter graph handles some events raised by filters that are not passed to the application. One example of this is the EC_REPAINT event notification. The default handling for this event is to repaint the video renderer's static images by pausing the filter graph. An application can override default handling for a specific event by calling the CancelDefaultHandling method with the event value as a parameter. The RestoreDefaultHandling method reinstates default handling for the specified event value. These methods have no effect on events that have no default handling.

If an error occurs during the transition to a running state on any filter, an error value is returned by the Run method. In this case, some filters within the graph might be running successfully. The filter graph leaves it up to the application to determine whether to stop the graph in case of an error. After the **Run** method has returned, event notifications report any additional errors. The EC_ERRORABORT and EC_USERABORT event notifications indicate that playback has probably stopped in the graph (certainly in the filter that reported it). Other errors and events indicate that it is still running. Note, however, that in all cases the graph remains in running mode until the application explicitly changes it to stopped or paused mode.

If the streams in the filter graph detect the end of the stream, the streams report this by using the EC_COMPLETE event notification. The filter graph asks filters if they can report **EC_COMPLETE** via *seekable renderers*.

A seekable renderer is a renderer that supports the IMediaPosition object from the filter and has only input pins, or is a renderer whose input pins report that they are rendered at that point. The filter graph detects seekable renderers. A seekable renderer reports EC_COMPLETE once when all seekable streams on that filter have reached the end of the stream.

The filter graph manager will then not pass EC_COMPLETE to the application until an **EC_COMPLETE** event notification has been received from each stream. For example, if a live camera stream is playing as the background to a video playing out of a file, the application will be notified about **EC_COMPLETE** when the video and audio streams from the file have come to the end of the stream, even though the live source is still playing. In this case, too, the filter graph remains in running mode until the application explicitly calls the Pause or Stop method.

The aggregation of EC_COMPLETE messages can be disabled by calling CancelDefaultHandling with **EC_COMPLETE** as the parameter. In this case, all **EC_COMPLETE** events raised by the filters will be passed directly to the application.

For a list of system-defined event notifications, see the GetEvent method.

Methods

Name	**Description**
CancelDefaultHandling	Cancels any default handling of the specified event by the filter graph.
GetEvent	Retrieves the next notification event.
GetEventHandle	Retrieves a handle to a manual-reset event that will be signaled.
RestoreDefaultHandling	Restores default handling for this event.
WaitForCompletion	Waits until the filter graph's operation has completed.

CancelDefaultHandling Method (IMediaEvent Object)

IMediaEvent Object

Cancels default handling of the specified event by the filter graph and ensures that the event is passed to the application.

objMediaEvent.**CancelDefaultHandling** *lEvCode*

Parts

objMediaEvent
Object expression that evaluates to an IMediaEvent object.

lEvCode
Long value specifying the event code which cancels default handling. The following values are valid.

EC_COMPLETE
EC_ERRORABORT
EC_PALETTE_CHANGED
EC_REPAINT
EC_USERABORT

Return Values

This method raises an error if the event does not have any default handling.

Remarks

Currently the filter graph manager only applies default handling to EC_COMPLETE and EC_REPAINT.

GetEvent Method (IMediaEvent Object)

IMediaEvent Object

Retrieves the next notification event.

objMediaEvent.**GetEvent** *lEventCode*, *lParam1*, *lParam2*, *msTimeout*

Parts

objMediaEvent
Object expression that evaluates to an IMediaEvent object.

lEventCode
Returns the next event notification. The following values are valid.
- EC_COMPLETE
- EC_ERRORABORT
- EC_PALETTE_CHANGED
- EC_REPAINT
- EC_USERABORT

lParam1
Long value specifying the first parameter of the event.

lParam2
Long value specifying the second parameter of the event.

msTimeout
Time, in milliseconds, to wait before assuming that there are no events. To return immediately, specify a time-out value of 0.

Remarks

This method removes the next event notification from the head of the queue and returns it. It waits up to *msTimeout* milliseconds if there are no events.

Previous Home Topic Contents Index Next

GetEventHandle Method (IMediaEvent Object)

IMediaEvent Object

Returns a handle to a manual-reset event that will be signaled as Long as there are event notifications to deliver.

objMediaEvent.**GetEventHandle** *hEvent*

Parts

objMediaEvent
Object expression that evaluates to an IMediaEvent object.
hEvent
Long value that will contain the event handle.

Remarks

The event can be passed to the Win32 WaitForMultipleObjects or MsgWaitForMultipleObjects function to wait for event notifications at the same time as other messages and events on a single thread.

RestoreDefaultHandling Method (IMediaEvent Object)

IMediaEvent Object

Reinstates the normal default handling by a filter graph for the specified event if there is one.

objMediaEvent.**RestoreDefaultHandling** *lEvCode*

Parts

objMediaEvent
  Object expression that evaluates to an IMediaEvent object.
lEvCode
  Event to restore. The following list contains system-defined event notifications.
  EC_COMPLETE
  EC_ERRORABORT
  EC_PALETTE_CHANGED
  EC_REPAINT
  EC_USERABORT

Remarks

Events that have default handling in place, such as EC_REPAINT, are not typically passed to the application.

WaitForCompletion Method (IMediaEvent Object)

IMediaEvent Object

Waits until the filter graph's operation has completed.

objMediaEvent.**WaitForCompletion** *msTimeout*, *EvCode*

Parts

objMediaEvent
  Object expression that evaluates to an IMediaEvent object.
msTimeout
  Long value that specifies the duration of the time-out, in milliseconds. To block indefinitely, pass –1. To return immediately, pass zero.
EvCode
  Long value that will contain the value of the event completion code, such as EC_COMPLETE. A value of zero indicates that the operation has not completed.

Remarks

This method performs the equivalent of blocking by calling GetEvent repeatedly until the method receives the EC_COMPLETE, EC_ERRORABORT, or EC_USERABORT event notification, or until it reaches the time-out period, whichever comes first.

If the time-out value is reached, the method returns a completion code value of 0 and raises run-time error 287, "Application-defined or object-defined error." When setting the *msTimeOut* parameter to anything other than −1 (infinite), you are likely to get this run-time error. You can handle this error with an **On Error Resume Next** statement and then check the value of *EvCode*, which will be 0 if a time-out has occurred or 1 (EC_COMPLETE) or greater if completion has occurred. The following example illustrates using this method with a time-out of 0.

```
Dim pME As IMediaEvent
Dim EventCode As Long
Set pME = pMC

On Error Resume Next

pME.WaitForCompletion 0, EventCode  'Return from WaitForCompletion immediately

Set pME = Nothing

If EventCode = 0 Then Exit Sub
```

To use an infinite time-out, check for a returned event code of 1 as in the following example.

```
Dim pME As IMediaEvent
Dim EventCode As Long
Set pME = pMC

pME.WaitForCompletion -1, EventCode 'Wait until EC_COMPLETE

If EventCode = 1 Then Exit Sub
```

When the method returns, the filter graph is still in a running state. This method assumes that the application is not making separate calls to the IMediaEvent object. This method fails if the graph is not in or transitioning into a running state.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaPosition Object

The filter graph exposes the **IMediaPosition** object if any of the filters within the graph are seekable (can seek to an arbitrary position in the stream). Filters will expose **IMediaPosition** if they can seek their data or if they connect to an output pin that represents a seekable stream.

Applications that communicate with the filter graph can call methods on this interface to set or retrieve properties such as the stream's duration, the start and stop times, the preroll time, the rate, and the current position. The filter graph uses these properties on seekable filters to control the playback of streams within the graph; where there are multiple streams, the filter graph sets them all to play in parallel, beginning at the same position, and will report the duration as being the duration of the longest stream. The parameters used in this interface are double values representing a fractional number of seconds. Internally, filters will store time to an accuracy of 100 nanoseconds.

Properties

Name	**Description**
CurrentPosition	Retrieves or sets the current position in terms of the total length of the media stream.
Duration	Retrieves the total duration of the media stream.
PrerollTime	Retrieves or sets the time prior to the start position at which nonrandom access devices should start rolling.
Rate	Retrieves or sets the playback rate, relative to normal playback of the media stream.
StopTime	Retrieves or sets the position within the media stream at which playback should stop.

Methods

Name	**Description**
CanSeekBackward	Retrieves whether or not you can seek backward in the current stream.
CanSeekForward	Retrieves whether or not you can seek forward in the current stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CurrentPosition Property (IMediaPosition Object)

IMediaPosition Object

Retrieves or sets the current position in terms of the total length of the media stream.

objMediaPosition.**CurrentPosition** [=*dblValue*]

Parts

objMediaPosition

Object expression that evaluates to an IMediaPosition object.

dblValue

Double value specifying the new current position.

Remarks

The **CurrentPosition** property represents the position that playback has reached. It is a value between zero and the duration (that is, it has been adjusted for rate and start time). If the graph is paused, this is the position at which it will restart.

This property can also be used to set the start time. When used in this way, the current position indicates a position between zero and the duration of the media at which playback should begin when the Run method is called.

When the filter graph is stopped or paused, this property returns the position at which playback will recommence. When the filter graph is running, the filter graph manager returns the position according to the reference clock. If an individual filter implements this, it should return the stream time of the sample it is currently processing (that is, the offset time from the beginning) when paused or running.

After stopping or pausing, a run command causes playback to begin at the current position. This will be where playback stopped or paused, unless the application changes the **CurrentPosition** call in the meantime.

If this method is called when the filter graph manager is running, the filter graph manager will pause the graph, run the method, and then issue a new run command.

Setting the current position when paused or stopped causes playback to resume from the new start position when the run command is issued.

The current position is applied before the rate and therefore is the position at typical playback speed.

Previous Home Topic Contents Index Next

Duration Property (IMediaPosition Object)

IMediaPosition Object

Retrieves the total duration of the media stream.

objMediaPosition.**Duration**

Parts

objMediaPosition

Object expression that evaluates to an IMediaPosition object.

Remarks

The duration assumes normal playback speed, and it is therefore unaffected by the rate.

Previous Home Topic Contents Index Next

PrerollTime Property (IMediaPosition Object)

IMediaPosition Object

Retrieves or sets the time prior to the start position that devices should start rolling.

objMediaPosition.**PrerollTime** [=*dblValue*]

Parts

objMediaPosition
Object expression that evaluates to an IMediaPosition object.
dblValue
Double value for the **PrerollTime** property.

Remarks

Preroll time is the time prior to the start position at which nonrandom access devices, such as tape players, should start rolling.

Previous Home Topic Contents Index Next

Rate Property (IMediaPosition Object)

IMediaPosition Object

Retrieves or sets the playback rate relative to normal playback speed.

objMediaPosition.**Rate** [= *dblValue*]

Parts

objMediaPosition
  Object expression that evaluates to an IMediaPosition object.
dblValue
  New value for the rate. **Double**.

Remarks

This property allows an application to speed up or slow down playback relative to the normal default playback speed. A rate of 1.0 indicates normal playback speed. Specifying 2.0 causes playback at twice the normal rate: a video created for 10 frames per second (fps) will be played back at 20 fps, if resources permit. Audio streams played back at above-normal speed increase the pitch rather than drop samples. A rate of 0.5 indicates half speed.

Previous Home Topic Contents Index Next

StopTime Property (IMediaPosition Object)

IMediaPosition Object

Retrieves or sets the time at which the media stream stops.

objMediaPosition.**StopTime** [= *dblValue*]

Parts

objMediaPosition
  Object expression that evaluates to an IMediaPosition object.
dblValue
  New value for the stop time.

Remarks

The stop time is a position between zero and the duration of the media at which playback should stop.

The stop position is applied before the rate and therefore is the position at typical playback speed.

Previous Home Topic Contents Index Next

CanSeekBackward Method (IMediaPosition Object)

IMediaPosition Object

Retrieves whether or not you can seek backward through the current stream.

objMediaPosition.**CanSeekBackward**

Parts

objMediaPosition
Object expression that evaluates to an IMediaPosition object.

Remarks

The return value is of type If the value is 0, you cannot seek through the current stream. If the value is -1, you can seek. If you cannot seek through the stream, you can still set the stream's current position with CurrentPosition.

This method can return only -1 (True) with media streams that consist of a single media type, such as a WAV file or video-only MPEG file.

Previous Home Topic Contents Index Next

CanSeekForward Method (IMediaPosition Object)

IMediaPosition Object

Retrieves whether or not you can seek forward through the current stream.

objMediaPosition.**CanSeekForward**

Parts

objMediaPosition
Object expression that evaluates to an IMediaPosition object.

Remarks

The return value is of type If the value is 0, you cannot seek through the current stream. If the value is -1, you can seek. If you cannot seek through the stream, you can still set the stream's current position with CurrentPosition.

This method can return only -1 (True) with media streams that consist of a single media type, such as a WAV file or video-only MPEG file.

Previous Home Topic Contents Index Next

IMediaTypeInfo Object

The filter graph manager exposes **IMediaTypeInfo**, which allows access to media type information such as the major GUIDs (globally unique identifiers).

Properties

Name	**Description**
Subtype	Retrieves the subtype GUID as a string.
Type	Retrieves the major type GUID as a string.

Previous Home Topic Contents Index Next

Subtype Property (IMediaTypeInfo Object)

IMediaTypeInfo Object

Retrieves the subtype GUID as a string.

objMediaTypeInfo.**Subtype**

Parts

objMediaTypeInfo
Object expression that evaluates to an IMediaTypeInfo object.

Return Values

Returns a string that contains the subtype GUID.

Previous Home Topic Contents Index Next

Type Property (IMediaTypeInfo Object)

IMediaTypeInfo Object

Retrieves the major type GUID as a string.

objMediaTypeInfo.**Type**

Parts

objMediaTypeInfo
Object expression that evaluates to an IMediaTypeInfo object.

Return Values

Returns a string containing the major type GUID.

IPinInfo Object

The filter graph manager exposes the **IPinInfo** object, which allows access to pin information such as the media type and the pin direction. It also allows control of pins, such as connecting, disconnecting, and rendering.

Properties

Name	**Description**
ConnectedTo	Retrieves the IPinInfo object for the pin to which the filter is connected.
ConnectionMediaType	Retrieves the media type on this connection.
Direction	Retrieves the pin direction.
FilterInfo	Retrieves the IFilterInfo object for the filter to which this pin belongs.
MediaTypes	Retrieves the IAMCollection object of preferred media types.
Name	Retrieves the name of this pin.
Pin	Retrieves the pin associated with this IPinInfo object.
PinID	Retrieves the pin identifier.

Methods

Name	Description
Connect	Connects to the following pin.
ConnectDirect	Connects directly to the specified pin.
ConnectWithType	Connects directly to the specified pin using the specified media type only.
Disconnect	Disconnects this pin and the corresponding connected pin from each other.
Render	Completes the filter graph downstream from this pin; that is, generates all filters and connections needed to generate a graph from this pin to the renderer filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Connect Method (IPinInfo Object)

IPinInfo Object

Connects two pins, using transform filters as necessary.

objPinInfo.**Connect** *objPin*

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.
objPin
IPinInfo object specifying the other pin in the connection.

Previous Home Topic Contents Index Next

ConnectDirect Method (IPinInfo Object)

IPinInfo Object

Connects the two pins directly, without using transform filters.

objPinInfo.**ConnectDirect** *objPin*

Parts

objPinInfo
: IPinInfo object specifying the other pin in the connection.

objPin
: IPinInfo object specifying the other pin in the connection.

Previous Home Topic Contents Index Next

ConnectedTo Property (IPinInfo Object)

IPinInfo Object

Retrieves the IPinInfo object for the pin to which the filter is connected.

Set *objConnectedPinInfo* **=** *objPinInfo*.**ConnectedTo**

Parts

objConnectedPinInfo
: IPinInfo object for the pin.

objPinInfo
: Object expression that evaluates to an IPinInfo object.

Previous Home Topic Contents Index Next

ConnectionMediaType Property (IPinInfo Object)

IPinInfo Object

Retrieves the IMediaTypeInfo object for this connection.

Set *objMediaTypeInfo* **=** *objPinInfo*.**ConnectionMediaType**

Parts

objMediaTypeInfo

IMediaTypeInfo object that will contain the new media type.

objPinInfo
: IPinInfo object specifying the source pin; the *objMediaTypeInfo* object copies this pin's media type.

Remarks

The IMediaTypeInfo object contains information about the major type and subtype that this pin supports.

ConnectWithType Method (IPinInfo Object)

IPinInfo Object

Connects directly to the specified pin using the specified media type.

objPinInfo.**ConnectWithType** *objPin*, *objMediaType*

Parts

objPinInfo
: Object expression that evaluates to an IPinInfo object.

objPin
: IPinInfo object specifying the other pin in the connection.

objMediaType
: Value specifying the media type used in the connection. The *objMediaType* object must support IPinInfo.

Remarks

The Connect and ConnectDirect methods can also be used to connect pins.

Direction Property (IPinInfo Object)

IPinInfo Object

Retrieves the pin direction.

objPinInfo.**Direction**

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

The value 0 indicates input, and the value 1 indicates output.

Disconnect Method (IPinInfo Object)

IPinInfo Object

Disconnects this pin and the corresponding connected pin from each other.

objPinInfo.**Disconnect**

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

Each pin has its own connection to the other. This method disconnects both pins.

FilterInfo Property (IPinInfo Object)

IPinInfo Object

Retrieves the IFilterInfo object for the filter that contains the current pin.

Set *objFilterInfo* = *objPinInfo*.**FilterInfo**

Parts

objFilterInfo
Object expression that evaluates to an IFilterInfo object.
objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

For more information, see the reference entry for IFilterInfo.

Previous Home Topic Contents Index Next

MediaTypes Property (IPinInfo Object)

IPinInfo Object

Retrieves the IAMCollection object that specifies the preferred media types for the current pin.

Set *objCollection* = *objPinInfo*.**MediaTypes**

Parts

objCollection
IAMCollection object that is returned.
objPinInfo
Object expression that evaluates to an IPinInfo object.

Previous Home Topic Contents Index Next

Name Property (IPinInfo Object)

IPinInfo Object

Retrieves the name of this pin.

objPinInfo.**Name**

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.

Pin Property (IPinInfo Object)

IPinInfo Object

Returns the IPin object.

Set *objPin* = *objPinInfo*.**Pin**

Parts

objPin
IPinInfo object that will contain the retrieved pin.
objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

No IPin programmable object interface is offered for Visual Basic® programmers. This property is provided for use with DLLs that take an **IPin** parameter, and for use as a parameter in calls to the IPinInfo object's Connect, ConnectDirect, and ConnectWithType methods.

PinID Property (IPinInfo Object)

IPinInfo Object

Retrieves the pin identifier.

objPinInfo.**PinID**

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

You can pass the **PinID** as a parameter to the FindPin method of the IFilterInfo object.

Previous Home Topic Contents Index Next

Render Method (IPinInfo Object)

IPinInfo Object

Builds a filter graph capable of rendering the media type belonging to this source filter output pin, adding all transform and rendering filters and establishing all connections that are needed.

objPinInfo.**Render**

Parts

objPinInfo
Object expression that evaluates to an IPinInfo object.

Remarks

This method establishes only those connections that are needed downstream from the specified pin; that is, in the direction of the renderer filter. It differs from the RenderFile method, which establishes a complete filter graph for all pins.

Previous Home Topic Contents Index Next

IRegFilterInfo Object

The **IRegFilterInfo** object provides information about filters registered on this computer.

Properties

Name	Description
Name	Retrieves the name of the filter.

Methods

Name	Description
Filter	Creates an instance of this filter, adds it to the graph, and returns an IFilterInfo object for the newly added filter.

Filter Method (IRegFilterInfo Object)

IRegFilterInfo Object

Creates an instance of this filter and adds it to the graph.

objRegFilterInfo.**Filter** *objFilterInfo*

Parts

objRegFilterInfo
 Object expression that evaluates to an IRegFilterInfo object.
objFilterInfo
 IFilterInfo object specifying the filter to add.

Name Property (IRegFilterInfo Object)

IRegFilterInfo Object

Retrieves the name of the filter.

objRegFilterInfo.**Name**

Parts

objRegFilterInfo
Object expression that evaluates to an IRegFilterInfo object.

Return Values

Returns the name of the registered filter object.

Remarks

The vendor specifies the filter name.

Previous Home Topic Contents Index Next

IVideoWindow Object

The **IVideoWindow** object supports the video window properties of a video renderer. It is a dual interface, accessible through Visual Basic® and Visual C++®, that controls a generic video window. Generally, this is a video renderer that draws video into a window on the display.

The **IVideoWindow** object supports both properties and methods. When operations require several properties to be changed simultaneously; methods are provided that allow a number of related properties to be changed simultaneously. For example, setting the position and size of the window can be done by four individual property calls or by the single method SetWindowPosition.

The video renderer must be connected to use the **IVideoWindow** properties and methods.

Properties

Name	**Description**
AutoShow	Retrieves or sets the flag that specifies if the window will be automatically shown on the first state change.
BackgroundPalette	Retrieves or sets the flag that indicates whether the renderer realizes its palette in the background.
BorderColor	Retrieves or sets the border color for the video window.
Caption	Retrieves or sets the text caption on the playback window.
FullScreenMode	Retrieves or sets the type of full-screen mode.
Height	Retrieves or sets the height of the video window.
Left	Retrieves or sets the x-axis coordinate for the video window.

MessageDrain	Retrieves or sets a window handle that is used by the video window to receive or post messages.
NotifyOwnerMessage	Forwards messages that have been received by a parent window to a child window owned by a filter.
Owner	Retrieves or sets the owning parent window for the video playback window.
Top	Retrieves or sets the y-axis coordinates for the video window.
Visible	Retrieves or sets the visibility of the window.
Width	Retrieves or sets the width of the video window.
WindowState	Retrieves or sets the current window state (such as visible or minimized).
WindowStyle	Retrieves or sets the playback window style.
WindowStyleEx	Retrieves or sets the playback window's extended style bits.

Methods

Name	**Description**
GetMaxIdealImageSize	Gets the ideal maximum image size for the video image playback (client) area.
GetMinIdealImageSize	Gets the ideal minimum image size for the video image playback (client) area.
GetRestorePosition	Retrieves the restored video window size when maximizing and minimizing.
GetWindowPosition	Retrieves the video window position.
HideCursor	Hides the cursor.
IsCursorHidden	Retrieves whether the cursor is visible or not.
NotifyOwnerMessage	Passes palette and color change messages on the filter graph.
SetWindowForeground	Sets the video window as the foreground window and optionally gives it focus.
SetWindowPosition	Sets the video window position on the display.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AutoShow Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets information about whether the window will be automatically shown.

objVideoWindow.**AutoShow** [= *boolean*]

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.
boolean
    True means the window will be made visible when the state is changed.

Remarks

Many simple applications require a displayed window when a filter graph is set to the running state. This method is set on (-1) by default so that when the graph changes state to paused or running, the window will be made visible (it will also be set to be the foreground window). It will be shown on all subsequent state changes to paused or running unless the window is closed, in which case it will not be shown again until the renderer goes through a stopped state. If this property is set to off (0), the window will remain hidden until explicitly shown. You can do this by setting the Visible property.

BackgroundPalette Property (IVideoWindow Object)

IVideoWindow Object

Determines whether any palette required will be realized in the background.

objVideoWindow.**BackgroundPalette** [= *boolean*]

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.
boolean
    Flag that indicates whether the palette will be realized.

Remarks

If this is True (-1), the renderer realizes any required video palette in the background. This means that any colors the palette uses will change to their closest match in the display palette prior to drawing. This ensures that an application will not have its palette disturbed when playing a video. It does, however, impose severe performance penalties on the video and should not be used unless absolutely necessary. The default value for this property is False (0).

BorderColor Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the border color for the video window.

objVideoWindow.**BorderColor** [= *Color*]

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.
Color
    New border color.

Remarks

When a destination rectangle that is set differs from the window's visible client area, a border is exposed around the edge. This method allows an application to change the border color. It is set to black by default. Any nonsystem color passed in is converted to its closest match, according to the current palette, before being used (this is not an issue on true-color devices). Setting this causes the window border to be repainted in the new color automatically.

Caption Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the textual title string for the video window.

objVideoWindow.**Caption** [= *string*]

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.
string

New value for the window title caption.

Previous Home Topic Contents Index Next

FullScreenMode Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets a flag indicating whether to use full-screen mode for the video window.

objVideoWindow.**FullScreenMode** [= *boolean*]

Parts

objVideoWindow
Object expression that evaluates to an IVideoWindow object.

boolean
New flag value. True means use full-screen mode; False means do not use full-screen mode.

Previous Home Topic Contents Index Next

GetMaxIdealImageSize Method (IVideoWindow Object)

IVideoWindow Object

Retrieves the ideal maximum image size for the video image playback (client) area.

objVideoWindow.**GetMaxIdealImageSize** *Width***,** *Height*

Parts

objVideoWindow
Object expression that evaluates to an IVideoWindow object.

Width
Long value that will contain the image width.

Height
    Long value that will contain the image height.

GetMinIdealImageSize Method (IVideoWindow Object)

IVideoWindow Object

Gets the ideal minimum image size for the video image playback (client) area.

objVideoWindow.**GetMinIdealImageSize** *Width*, *Height*

Parts

objVideoWindow
    An object expression that evaluates to an IVideoWindow object.
Width
    Long value that will contain the image width.
Height
    Long value that will contain the image height.

GetRestorePosition Method (IVideoWindow Object)

IVideoWindow Object

Retrieves the size and position of the window when it is to be restored after being minimized.

objVideoWindow.**GetRestorePosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.

Left
: Returns the x-axis origin of the video window.

Top
: Returns the y-axis origin of the video window.

Width
: Returns the width of the video window.

Height
: Returns the height of the video window.

GetWindowPosition Method (IVideoWindow Object)

IVideoWindow Object

Retrieves the current window rectangle (not the client rectangle) in device coordinates.

objVideoWindow.**GetWindowPosition** *Left*, *Top*, *Width*, *Height*

Parts

objVideoWindow
: Object expression that evaluates to an IVideoWindow object.

Left
: Returns the x-axis origin of the window.

Top
: Returns the y-axis origin of the window.

Width
: Returns the width of the window.

Height
: Returns the height of the window.

Remarks

This method has the same effect as individually setting the Left, Top, Width, and Height properties.

Height Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the height of the video window.

objVideoWindow.**Height** [= *lValue*]

Parts

objVideoWindow
  Object expression that evaluates to an IVideoWindow object.
lValue
  New value for the vertical dimension of the video window.

Remarks

Changes to the **Height** property are independent of the Width property (the y-coordinate of the video window).

HideCursor Method (IVideoWindow Object)

IVideoWindow Object

Specifies whether the cursor is visible or hidden when it passes over the video playback window.

objVideoWindow.**HideCursor** *bValue*

Parts

objVideoWindow
  Object expression that evaluates to an IVideoWindow object.
bValue
  Boolean value that specifies whether the cursor is visible or not; True hides the cursor and False makes it visible.

Remarks

When you use full-screen playback, it is common to hide the cursor to avoid distracting the user.

The following code fragment shows how to use this method.

```
        Dim o_objVideoWindow As IVideoWindow            'Initialize the IVideoWindc
        Dim g_objMediaControl As IMediaControl          'Initialize the IMediaContr

        Set g_objVideoWindow = g_objMediaControl                'Assign the playbac
        g_objVideoWindow.HideCursor (True)                      'Hide the cursor
```

IsCursorHidden Method (IVideoWindow Object)

IVideoWindow Object

Retrieves whether the cursor is visible or not.

objVideoWindow.**IsCursorHidden** *bValue*

Parts

objVideoWindow
    Object expression that evaluates to an IVideoWindow object.
bValue
    Boolean value that will contain the cursor's state of visibility; True if the cursor is hidden and False if it is visible.

Remarks

When you use full-screen playback, it is common to hide the cursor to avoid distracting the user.

The following code fragment shows how to use this method.

```
        Dim o_objVideoWindow As IVideoWindow            'Initialize the IVideoWindc
        Dim g_objMediaControl As IMediaControl          'Initialize the IMediaContr
        Dim bTest As Boolean                                    'Initialize the stc

        Set g_objVideoWindow = g_objMediaControl                'Assign the playbac
        g_objVideoWindow.IsCursorHidden (bTest)         'Check if the cursor is hic
```

Left Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the x-coordinate for the video window.

objVideoWindow.**Left** [= *lValue*]

Parts

objVideoWindow
  Object expression that evaluates to an IVideoWindow object.
lValue
  New value for the x-axis coordinate.

Remarks

Calling this method does not affect the width of the video window.

MessageDrain Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the window handle of the window that receives posted messages for the video window.

objVideoWindow.**MessageDrain** [= *handle*]

Parts

objVideoWindow
  Object expression that evaluates to an IVideoWindow object.
handle
  New value for the window handle.

NotifyOwnerMessage Method (IVideoWindow Object)

IVideoWindow Object

Forwards messages that have been received by a parent window to a child window owned by a filter.

objVideoWindow.**NotifyOwnerMessage** *hwnd*, *uMsg*, *wParam*, *lParam*

Parts

objVideoWindow
 Object expression that evaluates to an IVideoWindow object.
hwnd
 Window handle.
uMsg
 Message being sent.
wParam
 Message's *wParam*.
lParam
 Message's *lParam*.

Owner Property (IVideoWindow Object)

IVideoWindow Object

Retrieves or sets the owning parent for the video window.

objVideoWindow.**Owner** [= *handle*]