This sample cannot be rendered.		
VFW_E_SAMPLE_REJECTED_EOS	0x8004022C	556
This sample cannot be rendered because the end of the stream has been reached.		
VFW_E_DUPLICATE_NAME	0x8004022D	557
An attempt to add a filter with a duplicate name failed.		
VFW_S_DUPLICATE_NAME	0x0004022D	557
An attempt to add a filter with a duplicate name succeeded with a modified name.		
VFW_E_TIMEOUT	0x8004022E	558
A time-out has expired.		
VFW_E_INVALID_FILE_FORMAT	0x8004022F	559
The file format is invalid.		
VFW_E_ENUM_OUT_OF_RANGE	0x80040230	560
The list has already been exhausted.		
VFW_E_CIRCULAR_GRAPH	0x80040231	561
The filter graph is circular.		
VFW_E_NOT_ALLOWED_TO_SAVE	0x80040232	562
Updates are not allowed in this state.		
VFW_E_TIME_ALREADY_PASSED	0x80040233	563
An attempt was made to queue a command for a time in the past.		
VFW_E_ALREADY_CANCELLED	0x80040234	564
The queued command was already canceled.		
VFW_E_CORRUPT_GRAPH_FILE	0x80040235	565
Cannot render the file because it is corrupt.		
VFW_E_ADVISE_ALREADY_SET	0x80040236	566
An IOverlay advise link already exists.		
VFW_S_STATE_INTERMEDIATE	0x00040237	567
The state transition is not complete.		
VFW_E_NO_MODEX_AVAILABLE	0x80040238	568
No full-screen modes are available.		
VFW_E_NO_ADVISE_SET	0x80040239	569
This advise cannot be canceled because it was not successfully set.		
VFW_E_NO_FULLSCREEN	0x8004023A	570
Full-screen mode is not available.		
VFW_E_IN_FULLSCREEN_MODE	0x8004023B	571
Cannot call IVideoWindow methods while in full-screen mode.		
VFW_E_UNKNOWN_FILE_TYPE	0x80040240	576
The media type of this file is not recognized.		
VFW_E_CANNOT_LOAD_SOURCE_FILTER	0x80040241	577
The source filter for this file could not be loaded.		
VFW_S_PARTIAL_RENDER	0x00040242	578
Some of the streams in this movie are in an unsupported format.		
VFW_E_FILE_TOO_SHORT	0x80040243	579

A file appeared to be incomplete.		
VFW_E_INVALID_FILE_VERSION	0x80040244	580
The file's version number is invalid.		
VFW_S_SOME_DATA_IGNORED	0x00040245	581
The file contained some property settings that were not used.		
VFW_S_CONNECTIONS_DEFERRED	0x00040246	582
Some connections failed and were deferred.		
VFW_E_INVALID_CLSID	0x80040247	583
This file is corrupt: it contains an invalid class identifier.		
VFW_E_INVALID_MEDIA_TYPE	0x80040248	584
This file is corrupt: it contains an invalid media type.		
VFW_E_SAMPLE_TIME_NOT_SET	0x80040249	585
No time stamp has been set for this sample.		
VFW_S_RESOURCE_NOT_NEEDED	0x00040250	592
The resource specified is no longer needed.		
VFW_E_MEDIA_TIME_NOT_SET	0x80040251	593
No media time stamp was set for this sample.		
VFW_E_NO_TIME_FORMAT_SET	0x80040252	594
No media time format was selected.		
VFW_E_MONO_AUDIO_HW	0x80040253	595
Cannot change balance because audio device is mono only.		
VFW_S_MEDIA_TYPE_IGNORED	0x00040254	596
Could not connect with the media type in the persistent graph.		
VFW_E_NO_DECOMPRESSOR	0x80040255	597
Cannot play back the video stream: could not find a suitable decompressor.		
VFW_E_NO_AUDIO_HARDWARE	0x80040256	598
Cannot play back the audio stream: no audio hardware is available, or the hardware is not supported.		
VFW_S_VIDEO_NOT_RENDERED	0x00040257	599
Cannot play back the video stream: could not find a suitable renderer.		
VFW_S_AUDIO_NOT_RENDERED	0x00040258	600
Cannot play back the audio stream: could not find a suitable renderer.		
VFW_E_RPZA	0x80040259	601
Cannot play back the video stream: format 'RPZA' is not supported.		
VFW_S_RPZA	0x0004025A	602
Cannot play back the video stream: format 'RPZA' is not supported.		
VFW_E_PROCESSOR_NOT_SUITABLE	0x8004025B	603
DirectShow cannot play MPEG movies on this processor.		
VFW_E_UNSUPPORTED_AUDIO	0x8004025C	604
Cannot play back the audio stream: the audio format is not supported.		

VFW_E_UNSUPPORTED_VIDEO	0x8004025D	605	Cannot play back the video stream: the video format is not supported.
VFW_E_MPEG_NOT_CONSTRAINED	0x8004025E	606	DirectShow cannot play this video stream because it falls outside the constrained standard.
VFW_E_NOT_IN_GRAPH	0x8004025F	607	Cannot perform the requested function on an object that is not in the filter graph.
VFW_S_ESTIMATED	0x00040260	608	The value returned had to be estimated. Its accuracy can't be guaranteed.
VFW_E_NO_TIME_FORMAT	0x80040261	609	Cannot get or set time-related information on an object that is using a time format of
VFW_E_READ_ONLY	0x80040262	610	Could not make the connection because the stream is read-only and the filter alters the data.
VFW_S_RESERVED	0x00040263	611	This success code is reserved for internal purposes within DirectShow.
VFW_E_BUFFER_UNDERFLOW	0x80040264	612	The buffer is not full enough.
VFW_E_UNSUPPORTED_STREAM	0x80040265	613	Cannot play back the file: the format is not supported.
VFW_E_NO_TRANSPORT	0x80040266	614	Pins cannot connect because they don't support the same transport.
VFW_S_STREAM_OFF	0x00040267	615	The stream was turned off.
VFW_S_CANT_CUE	0x00040268	616	The graph can't be cued because it lacks data or contains corrupt data.
VFW_E_BAD_VIDEOCD	0x80040269	617	The Video CD can't be read correctly by the device or is the data is corrupt.
VFW_S_NO_STOP_TIME	0x80040270	618	The sample had a start time but not a stop time. In this case, the stop time returned is set to the start time plus one. The IMediaSample::GetTime method can return this success code.
VFW_E_OUT_OF_VIDEO_MEMORY	0x80040271	619	There is not enough video memory at this display resolution and number of colors. Reducing resolution might help.
VFW_E_VP_NEGOTIATION_FAILED	0x80040272	620	The video port connection negotiation process has failed.
VFW_E_DDRAW_CAPS_NOT_SUITABLE	0x80040273	621	

Either Microsoft DirectDraw® has not been installed or the video card capabilities are not suitable. Make sure the display is not in 16-color mode.		
VFW_E_NO_VP_HARDWARE	0x80040274	622
No video port hardware is available, or the hardware is not responding.		
VFW_E_NO_CAPTURE_HARDWARE	0x80040275	623
No Capture hardware is available, or the hardware is not responding.		
VFW_E_DVD_OPERATION_INHIBITED	0x80040276	624
This user operation is inhibited by DVD content at this time.		
VFW_E_DVD_INVALIDDOMAIN	0x80040277	625
This operation is not permitted in the current domain.		
VFW_E_DVD_NO_BUTTON	0x80040278	626
Requested button is not available.		
VFW_E_DVD_GRAPHNOTREADY	0x80040279	627
DVD-Video playback graph has not been built yet.		
VFW_E_DVD_RENDERFAIL	0x8004027a	628
DVD-Video playback graph building failed.		
VFW_E_DVD_DECNOTENOUGH	0x8004027b	629
DVD-Video playback graph could not be built due to insufficient decoders.		
CTL_E_CANTSAVEFILETOTEMP	0x800A02DF	735
CTL_E_SEARCHTEXTNOTFOUND	0x800A02E8	744
CTL_E_REPLACEMENTSTOOLONG	0x800A02EA	746
VFW_E_BAD_KEY	0x800403F2	1010
A registry entry is corrupt.		

Previous Home Topic Contents Index Next

Filters and Samples

This topic contains a brief description of the filters and sample applications shipped with DirectShow. The filters are supplied as binary code only and are available through the Filter Graph Editor. The samples include source code and demonstrate how to write DirectShow filters and applications. You can use these filters and samples as they are or modify them for your own applications.

- DirectShow Filters
- DirectShow Samples

DirectShow Filters

Microsoft® DirectShow™ provides filters and samples as part of the DirectShow Software Development Kit (SDK). A filter is supplied as binary code only, and is one of the filters listed in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu. Filters are described in this section.

Samples include source code. Some samples are filters and some are applications. Some of the sample filters are registered and appear in the Filter Graph Editor. Other sample filters must be built and registered before they will appear in the Filter Graph Editor. Sample filters (and sample applications) are described in DirectShow Samples.

The DirectShow SDK provides the following filters:

- ACM Audio Compressor
- Analog Video Crossbar
- Audio Capture
- Audio Renderer
- AVI Compressor
- AVI Decompressor
- AVI Draw
- AVI MUX
- AVI Splitter
- AVI/WAV File Source
- Color Space Converter
- Cutlist File Source
- DSound Audio Renderer
- DV Muxer
- DV Splitter
- DV Video Decoder
- DV Video Encoder
- DVD Navigator
- File Source (Async)
- File Source (URL)
- File Stream Renderer
- File Writer
- Full Screen Renderer
- 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
- Indeo 5.0 Video Progressive Download Sources
- Internal Script Command Renderer

- Line 21 Decoder
- Lyric Parser
- MIDI Parser
- MIDI Renderer
- MPEG Audio Decoder
- MPEG Video Decoder
- MPEG-1 Stream Splitter
- Multi-File Parser
- Overlay Mixer
- QuickTime Decompressor
- QuickTime Movie Parser
- SAMI (CC) Parser
- TrueMotion 2.0 Decompressor
- TV Audio
- TV Tuner
- VFW Video Capture
- VGA 16 Color Ditherer
- Video Renderer
- WAVE Parser
- WDM Video Capture

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ACM Audio Compressor

The ACM Audio Compressor filter acts as a container for the Audio Compression Manager (ACM), integrating the ACM with the DirectShow architecture. It supports the IAMStreamConfig interface, which lets you control audio capture or compression information from a filter's output pin.

Previous Home Topic Contents Index Next

Analog Video Crossbar

The WDM-based Analog Video Crossbar filter can select or route analog video. The single output stream represents a hardware path for analog baseband video. One of the input pins comes from a TV Tuner (the TV Tuner Filter). Other input pins support video streams.

The Analog Video Crossbar filter exposes the IAMCrossbar interface, which is used for routine source selection.

This filter is available through Windows 98 and Windows NT 5.

Previous Home Topic Contents Index Next

Audio Capture

The Audio Capture (WaveIn) filter is installed with the DirectShow run time. If you have audio capture hardware, this filter is available in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, under the Audio Capture Source category.

Unlike ordinary DirectShow filters, special category filters, such as Audio Capture Source filters, can work with more than one device. When DirectShow is installed, it will look for devices installed on your computer that work with the special category filter and list the options in that category. For example, in the Audio Capture Source filter category, DirectShow will list all the audio capture cards (which includes sound cards with microphone inputs) installed on the system. You then need to choose which device to use.

The audio capture filter has one capture output pin and several input pins (one for each type of input on the card, such as LineIn, Mic, CD, and MIDI).

The filter and its input pins each support IAMAudioInputMixer.

The output pin supports IAMStreamConfig.

This filter enables your sound capture card(s). If you have a sound card, it will be listed under in the Filter Graph Editor's list of filters in the Audio Capture Source category, with a name indicating the sound card it is enabling.

Previous Home Topic Contents Index Next

Audio Renderer

The Audio Renderer filter is a generic audio rendering filter that you can connect to the output of any of the following filters, if they contain WAV audio: File Source (Async), File Source (URL), MPEG-1 Stream Splitter, AVI Splitter, WAVE Parser, or any audio transform filter. This filter does not check the audio stream's subtype; the WAVEFORMAT or WAVEFORMATEX structure passed in the format block contains the information needed to connect to this filter.

The filter's property sheet contains the following:

Tab	**Property**	**Values**
Audio Input Pin (rendered)	Preferred Media Types	Lists the major type, subtype, and format.
Audio Renderer	**wFormatTag**	Waveform-audio format type. Many compression algorithms have registered format tags; the Mmreg.h header file contains a complete list of these format tags.
	nChannels	Number of channels in the waveform-audio data. Monaural data uses one channel and stereo data uses two channels.
	nSamplePerSec	Rate, in samples per second (hertz), at which each channel should play or record. If **wFormatTag** is WAVE_FORMAT_PCM, then common values for **nSamplePerSec** are 8.0 kHz, 11.025 kHz, 22.05 kHz, and 44.1 kHz. For non-PCM formats, you must compute this member according to the manufacturer's format specification.
	nAvgBytesPerSec	Required average data-transfer rate, in bytes per second, for the format tag. If **wFormatTag** is WAVE_FORMAT_PCM, then **nAvgBytesPerSec** should equal the product of **nSamplesPerSec** and **nBlockAlign**. For formats other than pulse code modulation (PCM), you must compute this member according to the manufacturer's format specification. Playback and record software can estimate buffer sizes by using the **nAvgBytesPerSec** member.
	nBlockAlign	Block alignment, in bytes. The block alignment is the smallest unit of data for the **wFormatTag** format type. If **wFormatTag** is WAVE_FORMAT_PCM, then **nBlockAlign** should equal **nChannels**. For non-PCM formats, this member must be computed according to the manufacturer's format specification. Playback and record software must process a multiple of **nBlockAlign** bytes of data at a time. Data written and read from a device must always start at the beginning of a block. For example, it is illegal to start playback of PCM data in the middle of a sample (that is, on a non-block-aligned boundary).
	Rate	Specifies the rendering rate for this file. This value acts as a multiplier; 1.0 represents the authored speed. Read-only.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AVI Compressor

The AVI compressor filter has one input pin and one output pin.

The filter supports the IAMVfwCompressDialogs interface.

The output pin supports the IAMStreamConfig and IAMVideoCompression interfaces.

AVI Decompressor

The AVI Decompressor filter decompresses AVI input and generates suitable output for a video rendering filter or an intervening video transform filter.

AVI Draw

This filter handles IOverlay interface support for an audio-video interleaved (AVI) video data stream. The AVI Draw filter's input pin connects to a video data output pin from the AVI Splitter filter. The AVI Draw output pin must connect to a filter that supports overlays; currently, the Video Renderer filter is the only such filter included with the DirectShow SDK.

DirectShow typically uses this filter to play back motion-JPEG (MJPEG) compressed AVI files on a computer with hardware support for MJPEG playback. In a general sense, this filter negotiates connections between specific hardware devices and AVI video streams that rely on those hardware devices, such as video capture and playback cards. This filter is not needed for playback or processing of streams that are hardware-independent.

AVI MUX

The AVI MUX filter takes the input from one or more media streams, converts the data if necessary, and combines it to produce a single output data stream in audio-video interleaved (AVI) format. This filter's output pin must be connected to the File Writer filter.

The AVI MUX's default behavior produces an AVI 1.0 format interleaved data stream. Use the IConfigAviMux interface to change the compatibility index.

The filter's property sheet contains the following:

Tab	**Property**	**Values**
Interleaving	Interleaving	Controls the level of interleaving between the source media files; you can specify the level of interleaving: none, capture, or full.
	Interleaving Parameters	Sets the audio preroll and interleaving frequency (measured in milliseconds).
	Throughput Statistics	Displays the number of dropped frames on the output data stream.
AVI Specific	Capture Drift: master stream	Enable and disable Capture Drift on the specified master stream.
AVI Out	Preferred Media Types	Displays the major type, subtype, and format of the output data stream.
Input 0X	Preferred Media Types	Displays the major type, subtype, and format of the specified input data stream.

Previous Home Topic Contents Index Next

AVI Splitter

The AVI Splitter filter parses AVI-compressed video data and splits it into the component data streams. It can be connected to the File Source (async), File Source (URL), or AVI/WAV File Source filter supplied with DirectShow, or a third-party filter that delivers a compressed AVI stream and can interface with an asynchronous parser.

Previous Home Topic Contents Index Next

AVI/WAV File Source

The AVI/WAV File Source filter reads AVI and WAV source files and generates the appropriate output pins for the file type.

For WAV files, the filter creates an audio output pin, which produces an audio stream that can be connected to an audio rendering filter or intervening audio transform filter.

For AVI files, the filter creates a video output pin, which produces a compressed AVI stream suitable for the AVI codec filter, and an audio output pin, which produces an audio stream suitable for an audio rendering filter or an intervening audio transform filter.

Previous Home Topic Contents Index Next

Color Space Converter

This transform filter converts from one RGB color type to another, such as between 24-bit and 8-bit RGB color. You can use this filter to convert to a color space used by the video rendering filter, such as from the AVI Decompressor filter to the Full Screen Renderer filter.

Previous Home Topic Contents Index Next

Cutlist File Source

This source filter can output different pieces of different .avi or .wav files seamlessly and efficiently. It can be used for nonlinear video editing and for audio editing.

Using the CutListGraphBuilder object, the SimpleCutList object, and the VideoFileClip and AudioFileClip objects, an application can build a cutlist out of pieces of .avi and .wav, and use the DirectShow Cutlist Source Filter to play it.

Previous Home Topic Contents Index Next

DSound Audio Renderer

The DSound Audio Renderer filter is a generic audio rendering filter that you can connect to the output of any of the following filters, if they contain WAV audio: File Source (Async), File Source (URL), MPEG-1 Stream Splitter, AVI Splitter, WAVE Parser, or any audio transform filter. In addition to its basic sound-rendering capabilities, this filter can process Microsoft DirectX® DirectSound® API calls; use the IAMDirectSound methods to set and retrieve the window that will handle the sound playback.

Note that this filter does not check the subtype of the audio stream; the WAVEFORMAT or WAVEFORMATEX structure passed in the format block contains the information needed to connect to this filter. The Audio Renderer is the default audio rendering filter for DirectShow; to use the **DSound Audio Renderer** filter instead, you must insert it into the filter graph before rendering the media file.

The filter's property sheet contains the following:

Tab	**Property**	**Values**
Audio Input pin (rendered)	Preferred Media Types	Lists the major type, subtype, and format
Audio Renderer	**wFormatTag**	Waveform-audio format type. There are many compression algorithms with registered format tags. You can find a complete list of format tags in the Mmreg.h header file.
	nChannels	Number of channels in the waveform-audio data. Monaural data uses one channel and stereo data uses two channels.
	nSamplePerSec	Rate, in samples per second (hertz), at which each channel should play or record. If **wFormatTag** is WAVE_FORMAT_PCM, then common values for **nSamplesPerSec** are 8.0 kHz, 11.025 kHz, 22.05 kHz, and 44.1 kHz. For non-PCM formats, you must compute this member according to the manufacturer's format specification.
	nAvgBytesPerSec	Required average data-transfer rate, in bytes per second, for the format tag. If **wFormatTag** is WAVE_FORMAT_PCM, **nAvgBytesPerSec** should equal the product of **nSamplesPerSec** and **nBlockAlign**. For non-PCM formats, you must compute this member according to the manufacturer's format specification. Playback and record software can estimate buffer sizes by using the **nAvgBytesPerSec** member.
	nBlockAlign	Block alignment, in bytes. The block alignment is the minimum atomic unit of data for the **wFormatTag** format type. If **wFormatTag** is WAVE_FORMAT_PCM, **nBlockAlign** should equal the product of **nChannels** and **wBitsPerSample** divided by 8 (bits per byte). For non-PCM formats, you must compute this member according to the manufacturer's format specification. Playback and record software must process a multiple of **nBlockAlign** bytes of data at a time. Data written and read from a device must always start at the beginning of a block. For example, it is illegal to start playback of PCM data in the middle of a sample (that is, on a non-block-aligned boundary).
	Rate	This value represents the rate of audio playback, where 1.0 is the authored speed. This value is a multiplier; a value of 2.0 is twice the authored speed and 0.5 is half.

Previous Home Topic Contents Index Next

DV Muxer

The DV Muxer filter multiplexes compressed DV video with compressed DV audio.

The DV Muxer filter accepts the following input media types:

From the video input pin:
MEDIATYPE_Video
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: VIDEOINFO

From the audio input pin:
MEDIATYPE_Audio
MEDIASUBTYPE_wave
FORMAT: WAVEFORMATEX/NULL

The DV Muxer filter sends the following output media type:

From the video output pin:
MEDIATYPE_Interleaved(iavs)
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: DVINFO

To see a diagram of how to use the DV Muxer filter with the DV Video Encoder filter, see DV Video Encoder.

Previous Home Topic Contents Index Next

DV Splitter

The DV Splitter filter passes a DV stream to a downstream filter (DV Video Decoder and converts DV audio to PCM audio.

The DV Splitter filter accepts the following input media type:
MEDIATYPE_Interleaved(iavs)
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: DVINFO

The DV Splitter filter outputs the following output media types:

From the video output pin:
MEDIATYPE_Video
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: VIDEOINFO

From the audio output pin:
MEDIATYPE_Audio
MEDIASUBTYPE_wave
FORMAT: WAVEFORMATEX

To see a diagram of how to use the DV Splitter filter with the DV Video Decoder filter, see DV Video Decoder.

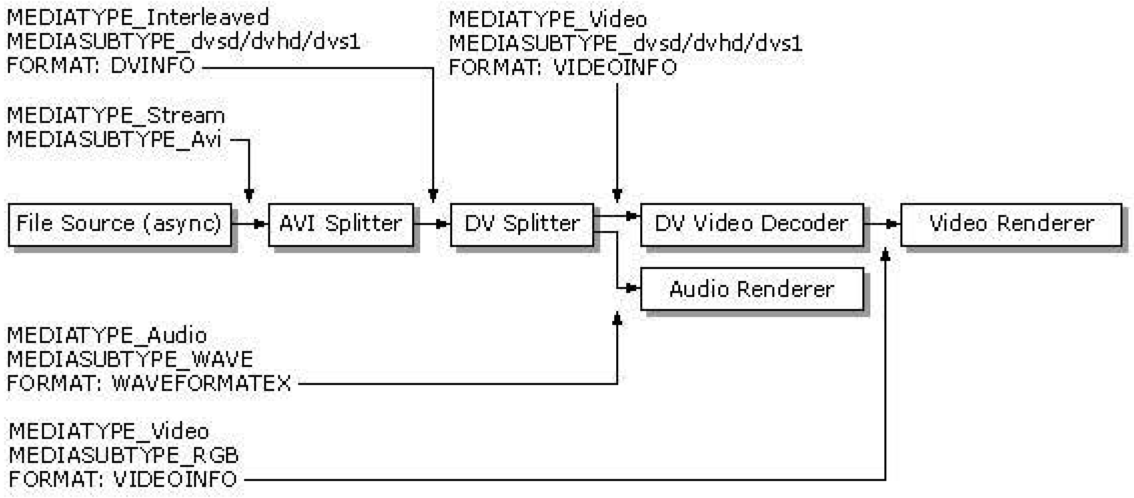
DV Video Decoder

The DV Video Decoder filter decodes a DV stream to uncompressed video.

The DV Video Decoder filter accepts the following input media type:
MEDIATYPE_Video
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: VIDEOINFO

The DV Video Decoder filter outputs the following output media type:
MEDIATYPE_Interleaved
MEDIASUBTYPE_RGB/YUYV/and other RGB and FOURCC subtypes
FORMAT: VIDEOINFO

The following illustration gives an example of how to use DV Video Decoder and DV Splitter filters.



Previous Home Topic Contents Index Next

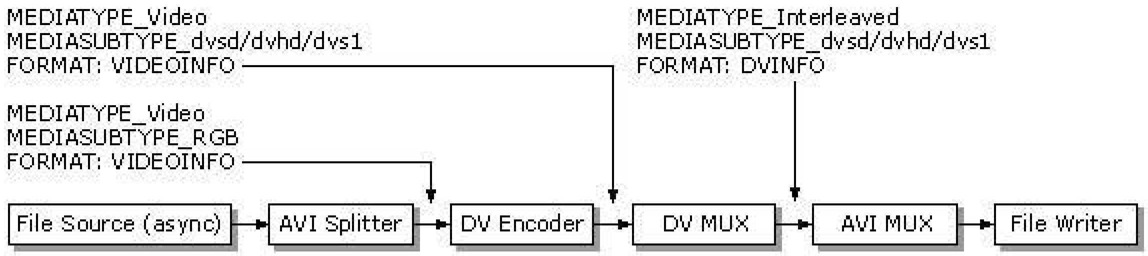
DV Video Encoder

The DV Video Encoder filter encodes uncompressed video into a compressed DV stream.

The DV Video Encoder filter accepts the following input media type:
MEDIATYPE_Video
MEDIASUBTYPE_RGB/YUYV/and other RGB and FOURCC subtypes
FORMAT: VIDEOINFO

The DV Video Encoder filter outputs the following output media type:
MEDIATYPE_Interleaved
MEDIASUBTYPE_dvsd/dvhd/dvsl
FORMAT: VIDEOINFO

The following illustration gives an example of how to use DV Video Encoder and DV Muxer filters.



DVD Navigator

The DVD Navigator filter is the source filter for a DVD-Video playback filter graph. It opens all necessary files in a DVD-Video volume, navigates through the linear DVD-Video .vob files, and parses the resulting MPEG-2 program stream, splitting the stream into three (video, audio, subpicture) output pins.

The DVD Navigator filter also implements the IDvdControl and IDvdInfo interfaces that enable a DVD playback application to control DVD-Video playback.

File Source (Async)

The File Source (async) filter is a generic asynchronous source filter that works with any source file whose media major type is stream. This includes AVI, MOV, MPEG, and WAV files. It requires the downstream filter to be a parser, such as the MPEG-1 Stream Splitter, the AVI Splitter, or the QuickTime Movie Parser.

File Source (URL)

The File Source (URL) filter is a generic asynchronous source filter that works with any source file that can be identified by a Uniform Resource Locator (URL) and whose media major type is stream. This includes AVI, MOV, MPEG, and WAV files. It requires the downstream filter to be a parser, such as the MPEG-1 Stream Splitter, the AVI Splitter, or the QuickTime Movie Parser.

File Stream Renderer

The File Stream Renderer filter renders streams with the major media type MEDIATYPE_File. The filter hooks up to output pins of type MEDIATYPE_File and renders a separate filename when the pin is rendered. The filename is contained in the **pFormat** field of the media type.

File Writer

The File Writer filter writes a stream of bits to disk. The output format matches the input format. The AVI MUX filter is currently the only filter to which the File Writer can connect, so it writes all data to the specified file in audio-video interleaved (AVI) format. You can create a new output file or specify an existing file; if the file already exists, it will be completely overwritten with the new data.

The time stamps on samples delivered to the file writer are byte offsets in the file. The input pin exposes the COM IStream interface and supports a subset of the interface. This enables the upstream filter to write data when the graph is stopped.

Full Screen Renderer

The Full Screen Renderer filter is a Mode X rendering filter that occupies the entire desktop when it displays video.

Full-screen rendering is enabled from the filter graph manager, which automatically switches between the filter graph's video renderer and the Full Screen Renderer when required; you need not include the Full Screen Renderer in the filter graph beforehand.

The filter's property sheet contains the following:

Tab	Property	Values
Quality	Frames played	Total frames in the video stream
	Frames dropped in renderer	Number of frames dropped during playback
	Average frame rate achieved	Average frames per second displayed during playback
	Jitter (std dev frame time) (mSec)	Standard deviation frame rate, in milliseconds
	Average sync offset (mSec)	Average synchronization offset, in milliseconds
	Std dev sync offset (mSec)	Standard deviation synchronization offset, in milliseconds
Input	Preferred Media Types	Lists the major type, subtype, and format

Previous Home Topic Contents Index Next

Internal Script Command Renderer

The Internal Script Command Renderer filter accepts data in with a major media type of MEDIATYPE_Text or MEDIATYPE_ScriptCommand, and, when it is time to render a particular piece of data, files an EC_OLE_EVENT event with the text or command as the parameters.

Previous Home Topic Contents Index Next

Indeo 4.3 Video Compression

The Indeo® 4.3 Video Compression filter compresses video data into the Indeo 4.3 format. This is typically used when writing out a video file for playback in, for example, a capture filter graph.

For more information about Intel Indeo Video, see the Intel Web site at http://developer.intel.com/ial/indeo .

Previous Home Topic Contents Index Next

Indeo 4.3 Video Decompression

The Indeo® 4.3 Video Decompression filter will decompress Indeo 4.3, 4.2, and 4.1 videos. This filter can be used during playback of Indeo 4.0 video files.

For more information about Intel Indeo Video, see the Intel Web site at:
http://developer.intel.com/ial/indeo

Indeo 5.0 Audio Decompression

The Indeo® 5.0 Audio Decompression filter decompresses Indeo 5.0 audio. This filter can be used during playback of Indeo 5.0 audio files.

For more information about Intel Indeo Audio, see the Intel Web site at
http://developer.intel.com/ial/indeo .

Indeo 5.0 Video Compression

The Indeo® 5.0 Video Compression filter compresses video data using the Indeo 5.0 algorithm. This is typically used in a video editing filter graph.

For more information about Intel Indeo Video, see the Intel Web site at
http://developer.intel.com/ial/indeo .

Indeo 5.0 Video Decompression

The Indeo® 5.0 Video Decompression filter decompresses video data using the Indeo 5.0 algorithm. This filter can be used during playback of Indeo 5.0 video files.

Previous Home Topic Contents Index Next

Indeo 5.0 Video Progressive Download Sources

Indeo® Progressive Download video files can be played as they download. The Indeo Video 5.0 Progressive Download Sources filter enables you to play these files. Lower resolution and lower-frame-rate video can be viewed almost immediately, while the display of the Progressive Download file continuously improves in both quality and frame. Indeo Audio Software works with the Indeo Video 5.0 Progressive Download Sources filter.

For more information about Intel Indeo Video, see the Intel Web site at http://developer.intel.com/ial/indeo .

Previous Home Topic Contents Index Next

Line 21 Decoder

The Line 21 Decoder filter takes Line 21 data (closed captions) and decodes it to produce a bitmap of media type MEDIATYPE_VIDEO. The output can then be rendered through a video renderer or mixed in a video mixer, such as the Overlay Mixer.

The Line 21 input data to the filter can be either of two forms: byte-pair data that comes on Line 21 of a video frame or feed, or DVD Line 21 data. For every group of pictures (GOP) in the DVD video stream, there can be a user data packet that has that particular GOP's header information and Line 21 data.

The Line 21 Decoder filter supports the IAMLine21Decoder and IBaseFilter interfaces externally, and the IAMovieSetup interface internally.

Previous Home Topic Contents Index Next

Lyric Parser

The Lyric Parser filter parses closed captioning information from simple text files. Also see SAMI (CC) Parser and the Multi-File Parser.

The Lyric Parser filter file format is as shown in the following example:

```
;LYRICS
0           Here's some text at the beginning
1000        This shows up after one second
5000        And this after four more seconds
10000       The end.
```

MIDI Parser

The MIDI Parser filter parses MIDI data from the File Source (Async) and File Source (URL) filters. The MIDI Renderer filter can render data from this filter.

MIDI Renderer

The MIDI Renderer filter renders MIDI data from the MIDI Parser filter.

MPEG Audio Decoder

The MPEG Audio Decoder filter decompresses MPEG-1 audio data. You usually connect it to the audio output of the MPEG-1 Stream Splitter filter. The MPEG Audio Decoder filter produces an audio stream suitable for input to an audio rendering filter or intervening audio transform filter.

MPEG Video Decoder

The MPEG Video Decoder filter decompresses MPEG-1 video data. Usually, its input comes from the MPEG-1 Stream Splitter filter and its output goes to a video rendering filter or an intervening video transform filter.

MPEG-1 Stream Splitter

The MPEG-1 Stream Splitter filter splits MPEG-1 data into separate audio and video streams. The upstream filter must be the File Source (Async) filter, File Source (URL) filter, or a compatible third-party asynchronous source filter. Its output is compressed video and audio data; you can connect the output to suitable decompressors, such as the MPEG Video Decoder filter and the MPEG Audio Decoder filter.

Multi-File Parser

The Multi-File Parser filter parses a simple file format that enables multiple actual file names to be specified as though they were one file. This assists you in combining video and text for closed captioning. Also see SAMI (CC) Parser and Lyric Parser.

Multi-File Parser files have the format shown in the following example:

```
;MULTI
http://server/share/video.mpg
http://server/share/captions.smi
```

Previous Home Topic Contents Index Next

Overlay Mixer

The Overlay Mixer filter provides video port playback support. The filter negotiates the parameters that control the video port with an upstream proxy filter that controls the video port driver. It then renders the video on the screen. It can also mix the video content with closed captions on a second pin, and can be extended to an arbitrary number of pins to add subpicture data and other video components.

Previous Home Topic Contents Index Next

QuickTime Decompressor

The QuickTime Decompressor filter decompresses files using the Apple® QuickTime® file format. This is typically used during play back of QuickTime files.

Previous Home Topic Contents Index Next

QuickTime Movie Parser

The QuickTime Movie Parser filter splits Apple® QuickTime® data into audio and video streams. The input pin connects to a source filter such as the File Source (Async) filter or the File Source (URL) filter. The Parser uses the Video For Windows (VFW) decompressor to decompress QuickTime files. If the Video for Windows compressor does not support the compression method, it cannot render the data. The filter creates one output pin for the video stream and one output pin for the audio stream.

Previous Home Topic Contents Index Next

SAMI (CC) Parser

The SAMI Closed-Caption Parser filter parses closed captioning information from SAMI-formatted files. Also see Lyric Parser.

The SAMI (Synchronized Accessible Media Interchange) interchange format is based on SGML/HTML. Every SAMI document should start with a <SAMI> tag and end with a </SAMI>tag. The document's class name is "Synchronized Accessible Media Interchange" and its file extension is .smi or .sami. At its most basic level, SAMI can be used as an intermediate interchange format for encoding closed captions in the Line-21 for NTSC, MPEG for DVDs, and IFE-ITV formats, and other similar formats.

The following is a sample SAMI document:

```
<SAMI>
<HEAD>
        <Title>President John F. Kennedy Speech</Title>
        <SAMIParam><!--
                Copyright="(C)Copyright 1997, Microsoft Corporation"
                Media="JF Kennedy.wav", none
                Length=73000
                CaptionMetrics=scaleable
                CaptionLineLength=180
                CaptionFontSize=12
                CaptionTextLines=3-->
      </SAMIParam>

        <STYLE TYPE="text/css"><!--
                P {margin-left: 29pt; margin-right: 29pt; font-size: 12pt;
                text-align: left; font-family: tahoma, arial, sans-serif;
                font-weight: normal; color: white; background-color: black;}

                TABLE {Width: "248pt" ;}

                .ENUSCC {Name: "English Captions"; lang: en-US-CC;}

                #Source {margin-bottom: -15pt; background-color: silver;
                        color: black; vertical-align: normal; font-size: 12pt;
                        font-family: tahoma, arial, sans-serif;
                        font-weight: normal;}

                #Youth {color: greenyellow; font-size: 18pt;}

                #BigPrint-1 {color: yellow; font-size: 24pt;}-->
        </STYLE>
</HEAD>

<BODY><TABLE>

        <SYNC Start=0>
                <P Class=ENUSCC ID=Source>Pres. John F. Kennedy
        <SYNC Start=10>
                <P Class=ENUSCC>Let the word go forth,
                        from this time and place to friend and foe
                        alike that the torch
        <SYNC Start=8800>
                <P Class=ENUSCC>has been passed to a new generations of Americans,
                        born in this century, tempered by war,
        <SYNC Start=19500>
                <P Class=ENUSCC>disciplined by a hard and bitter peace,
                        proud of our ancient heritage, and unwilling to witness
        <SYNC Start=28000>
                <P Class=ENUSCC>or permit the slow undoing of those human rights
                         to which this nation has always
        <SYNC Start=38000>
                <P Class=ENUSCC>been committed and to which we are
                        committed today at home and around the world.
        <SYNC Start=46000>
                <P Class=ENUSCC>Let every nation know,
                        whether it wishes us well or ill,
                        that we shall pay any price, bare any burden,
        <SYNC Start=61000>
                <P Class=ENUSCC>meet any hardship, support any friend,
                        oppose any foe, to ensure the survival and
                        success of liberty.
        <SYNC Start=73000>
                <P Class=ENUSCC ID=Source>End of:
                <P Class=ENUSCC>President John F. Kennedy Speech
</TABLE></BODY>
</SAMI>
```

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

TrueMotion 2.0 Decompressor

The TrueMotion Decompressor filter decompresses TrueMotion video. This filter will be used during playback of TrueMotion video files.

Previous Home Topic Contents Index Next

TV Audio

The TV Audio filter provides control of television audio decoding, stereo or monoaural selection, and secondary audio program (SAP) selection.

This filter is available through the Windows 98 and Windows NT 5.

TV Tuner

The TV Tuner filter selects an analog broadcast or cable channel to be viewed. The single output stream is a hardware path for analog baseband video. This output should be an input to the Analog Video Crossbar filter. The input pins include an input for cable and an antenna input.

The TV Tuner filter exposes the IAMTVTuner interface, which is used for channel selection.

This filter is available through the Windows 98 and Windows NT 5.

VFW Video Capture

The Video Capture filter is installed with the DirectShow run time. If you have video capture hardware that uses Video For Windows, this filter is available in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, under the Video Capture Sources.

Unlike ordinary DirectShow filters, special category filters, such as Video Capture Source filters, can work with more than one device. When DirectShow is installed, it will look for devices installed on your computer that work with the special category filter and list the options in that category. For example, in the Video Capture Source filter category, DirectShow will list all the video capture cards installed on the system. You then need to choose which device to use.

The Video for Windows Video Capture filter has two output pins called Capture and Preview.

The filter supports the IAMVfwCaptureDialogs interface.

The capture output pin supports the IAMStreamConfig, IAMVideoCompression, and IAMDroppedFrames interfaces.

VGA 16 Color Ditherer

The VGA 16 Color Ditherer filter converts from an RGB color type to a 4-bit color display. You can use this filter to convert to the color space used by video rendering filter, such as from the AVI Decompressor filter to the Full Screen Renderer filter.

Previous Home Topic Contents Index Next

Video Renderer

The Video Renderer filter is a generic video renderer that you can connect to any video transform filter that produces decompressed video data. This filter has its own plug-in distributor in the filter graph manager, which enables applications to set and retrieve properties on the filter by calling the corresponding interface methods on the filter graph manager. Most other DirectShow™ filters are not visible to applications in this manner.

The Video Renderer filter uses DirectX® methods, if the video card supports them. Full-screen rendering is enabled from the filter graph manager, which automatically switches between the Video Renderer and the Full Screen Renderer when appropriate; you need not include the Full Screen Renderer in the filter graph beforehand.

Previous Home Topic Contents Index Next

WAVE Parser

The WAVE Parser filter parses WAV-format audio data. The upstream filter must be the File Source (Async) filter, File Source (URL) filter, or a compatible third-party asynchronous source filter that contains WAV audio data. The output stream is uncompressed audio data, which you can connect directly to an audio rendering filter or to an intervening audio transform filter.

Previous Home Topic Contents Index Next

WDM Video Capture

The Video Capture filter is installed with the DirectShow run time. If you have video capture hardware that uses WDM (Windows Driver Model) and you have Windows 98 or Windows NT 5 installed, this filter is available in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, under the Video Capture Source category.

Unlike ordinary DirectShow filters, special category filters, such as Video Capture Source filters, can work with more than one device. When DirectShow is installed, it will look for devices installed on your computer that work with the special category filter and list the options in that category. For example, in the Video Capture Source filter category, DirectShow will list all the video capture cards installed on the system. You then need to choose which device to use.

The WDM Video Capture filter supports the interfaces and property sets of the WDM capture drivers. It exposes the **IAMAnalogVideoDecoder**, **IAMVideoProcAmp** and **IAMCameraControl** interfaces to applications. Depending on the underlying hardware, the filter exposes one or more output pins that output compressed video, uncompressed video, timecode, and closed captioning data. The output pins expose the IAMStreamControl interface, and in the case of compressed video, the IAMVideoCompression interface as well.

DirectShow Samples

This article provides the descriptions of the Microsoft® DirectShow™ samples that are part of the DirectShow Software Development Kit (SDK). These samples demonstrate how to write DirectShow filters and applications that use them.

Microsoft® DirectShow™ provides filters and samples as part of the DirectShow Software Development Kit (SDK). A filter is supplied as binary code only. Samples include source code. Some samples are filters and some are applications. Some of the sample filters are registered and appear in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu. Other sample filters must be built and registered before they will appear in the Filter Graph Editor.

The sample paths in this topic refer to the default sample directories created when you install the DirectX Media SDK.

The following sections describe the purpose and functionality of each sample filter or sample application.

Source Filters

- Async Sample (Asynchronous Reader Filter)
- Ball Sample (Bouncing Ball Filter)
- Synth Sample (Audio Synthesizer Filter)
- VidCap Sample (Video Capture Filter)

Transform Filters

- Contrast Sample (Video Contrast Filter)
- EzRGB24 Sample (Image Effect Filter)
- Gargle Sample (Gargle Filter)
- Inftee Sample (Infinite-Pin Tee Filter)
- MPGAudio Sample (MPEG Audio Decoder Filter)
- MPGVideo Sample (MPEG Video Decoder Filter)
- Nullip Sample (Null In Place Filter)
- Nullnull Sample (Minimal Null Filter)
- Vcrctrl Sample (VCR Control Filter)

Renderer Filters

- Dump Sample (Dump Filter)
- SampVid Sample (Video Renderer Filter)
- Scope Sample (Oscilloscope Filter)
- TextOut Sample (Text Display Filter)

Cutlist Sample Applications

- CLText Sample (Text Cutlist Application)
- Simplecl Sample (Cutlist Application)

C/C++-based Sample Applications

- AMCap Sample (DirectShow Capture Application)
- CPlay Sample (C/COM-based Media Player Application)
- Dvdsampl Sample (DVD Player Application)
- InWindow Sample (Window Playback Application)
- IPlay Sample (Indeo Player Application)
- MFCPlay Sample (C++/COM-based Media Player Application)
- MPEGProp Sample (MPEG Property Page Display Application)
- PlayFile Sample (Simple Playback Application)
- ShowStrm Sample (Multimedia Streaming Application)
- VidClip Sample (Video Editing Application)

Visual Basic-based Sample Applications

- Visual Basic-Based ActiveX Player
- Visual Basic-Based Filter Graph Builder
- Visual Basic-Based Filter Graph Player
- Visual Basic-Based Player

Miscellaneous Samples

- PID Sample (Plug-In Distributor Application)
- SampIOS Sample (IOStream Helper Library)

Additional Sample Information

- Structure of Comments in Sample Source Code
- Build Information

Sample Locations

The following table shows the directory location of each sample, assuming the default installation directory.

Sample	Directory location
AMCap Sample (DirectShow Capture Application)	DXMedia\Samples\Ds\Capture\
Async Sample (Asynchronous Reader Filter)	DXMedia\Samples\Ds\Async\
Ball Sample (Bouncing Ball Filter)	DXMedia\Samples\Ds\Ball\
CLText Sample (Text Cutlist Application)	DXMedia\Samples\Ds\Cutlist\Cltext
Contrast Sample (Video Contrast Filter)	DXMedia\Samples\Ds\Contrast\
CPlay Sample (C/COM-based Media Player Application)	DXMedia\Samples\Ds\Player\Cplay\
Dump Sample (Dump Filter)	DXMedia\Samples\Ds\Dump\

Dvdsampl Sample (DVD Player Application)	DXMedia\Samples\Ds\Dvdsampl\
EzRGB24 Sample (Image Effect Filter)	DXMedia\Samples\Ds\Ezrgb24\
Gargle Sample (Gargle Filter)	DXMedia\Samples\Ds\Gargle\
Inftee Sample (Infinite-Pin Tee Filter)	DXMedia\Samples\Ds\Inftee\
InWindow Sample (Window Playback Application)	DXMedia\Samples\Ds\Player\Inwindow\
IPlay Sample (Indeo Player Application)	DXMedia\Samples\Ds\Iplay\
MFCPlay Sample (C++/COM-based Media Player Application)	DXMedia\Samples\Ds\Player\Mfcplay\
MPGAudio Sample (MPEG Audio Decoder Filter)	DXMedia\Samples\Ds\Mpgaudio\
MPGVideo Sample (MPEG Video Decoder Filter)	DXMedia\Samples\Ds\Mpgvideo\
MPEGProp Sample (MPEG Property Page Display Application)	DXMedia\Samples\Ds\Player\Mpegprop\
Nullip Sample (Null In Place Filter)	DXMedia\Samples\Ds\Nullip\
Nullnull Sample (Minimal Null Filter)	DXMedia\Samples\Ds\Nullnull\
PID Sample (Plug-In Distributor Application)	DXMedia\Samples\Ds\pids\Iamovie\
PlayFile Sample (Simple Playback Application)	DXMedia\Samples\Ds\Player\Playfile\
SampIOS Sample (IOStream Helper Library)	DXMedia\Samples\Ds\Iostream\
SampVid Sample (Video Renderer Filter)	DXMedia\Samples\Ds\Sampvid\
Scope Sample (Oscilloscope Filter)	DXMedia\Samples\Ds\Scope\
ShowStrm Sample (Multimedia Streaming Application)	DXMedia\Samples\Ds\Showstrm\
Simplecl Sample (Cutlist Application)	DXMedia\Samples\Ds\Cutlist\Simplecl
Synth Sample (Audio Synthesizer Filter)	DXMedia\Samples\Ds\Synth\
TextOut Sample (Text Display Filter)	DXMedia\Samples\Ds\Textout\
Vcrctrl Sample (VCR Control Filter)	DXMedia\Samples\Ds\Vcrctrl\
VidCap Sample (Video Capture Filter)	DXMedia\Samples\Ds\Vidcap\
VidClip Sample (Video Editing Application)	DXMedia\Samples\Ds\Vidclip\
Visual Basic-Based ActiveX Player	DXMedia\Samples\Ds\Vb\Ocx\
Visual Basic-Based Filter Graph Builder	DXMedia\Samples\Ds\Vb\Builder\
Visual Basic-Based Filter Graph Player	DXMedia\Samples\Ds\Vb\Vbdemo\
Visual Basic-Based Player	DXMedia\Samples\Ds\Vb\Player\

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Structure of Comments in Sample Source Code

The source code for each sample generally includes a standard introduction in one of the source files. This introduction consists of a block of comments in one of the source files describing what the sample illustrates, what base classes it implements, and so on. To find the introduction to the sample, go to the .cpp file whose name matches the sample. (For samples written in C, the file name ends in .c instead.) The following template shows this introduction with standard headings illustrating the elements that might be included.

```
// What this sample illustrates
     [brief series of 1-line descriptions]
//
// Summary
     [paragraph on what's in the sample, what it does]
//
// Demonstration instructions
     [how to bring it up, for example under GraphEdt, and make it run]
//
// Implementation
     [introduction to the code, how it all works together]
//
// Known problems ("features not illustrated by this sample"):
     [optional]
//
// Files
     [file names and a half-line description of each]
//
// Base classes used (refer to the docs for a diagram of what they inherit):
     [list of base classes directly inherited from (summary of all files)]
```

Source Filters

This article details sample source filters. By definition, each of these source filter samples provides at least one output pin. (That is part of what makes them source filters as opposed to other kinds of filters.)

Contents of this article:

- Async Sample (Asynchronous Reader Filter)
- Ball Sample (Bouncing Ball Filter)
- Synth Sample (Audio Synthesizer Filter)
- VidCap Sample (Video Capture Filter)

Async Sample (Asynchronous Reader Filter)

The asynchronous reader sample filter, Async, shows how to implement "progressive download" in the DirectShow environment. Async implements the IAsyncReader interface, which is perhaps the most important aspect of asynchronous processing. It also implements the IFileSourceFilter interface and defines some helper classes to create filters that conform to the **IAsyncReader** interface.

The Memfile.exe application uses part of the Async sample code. This program reads a file into memory at a given rate (in kilobytes per second) and plays that file as it comes in.

This filter is not installed in the DirectShow run time and is not included in the list of DirectShow filters in the Filter Graph Editor. To use this filter, you must build and register it.

Sample Locations

Ball Sample (Bouncing Ball Filter)

The bouncing ball sample source filter, Ball, illustrates format negotiation. Ball also illustrates the use of the source filter base classes CSource and CSourceStream.

The code in Fball.h and Fball.cpp manages the filter interfaces. Those two files contain approximately the minimum code required for a source filter. The Ball.h and Ball.cpp files contain the code that bounces the ball.

This filter has a single output pin, which provides a video stream that shows a ball bouncing around in the frame. The Ball filter also accepts quality-management requests from the downstream graph, which illustrates a simple quality-management strategy. This filter implements the IQualityControl interface for that purpose.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Synth Sample (Audio Synthesizer Filter)

The audio synthesizer sample filter, Synth, demonstrates usage of the CSource and CSourceStream classes in an environment that derives its source from audio. You can use the Synth filter as a source filter to synthesize audio waveforms such as sine waveforms, square waveforms, sawtooth waveforms, and swept frequency waveforms.

The Synth source filter enables the user to set the waveform, frequency, number of channels, and other properties. The user can set these properties through the property page. To set either the upper or lower endpoint of the swept frequency range, hold down SHIFT while adjusting the frequency slider. The property page enables only the controls that affect the audio format — for example, channels, bits per sample, and sampling frequency — while the filter is in a stopped state.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

VidCap Sample (Video Capture Filter)

The VidCap sample video capture filter shows you how to handle special category filters, how to implement the COM interfaces for video capture, and how to implement miscellaneous other requirements for video capture.

The VidCap sample filter is not installed with the DirectShow run time. You must install it by calling Regsvr32.exe for VidCap.ax. The sample video capture filter is not part of the run time, but is part of the SDK.

A video capture filter with the same capability as VidCap is installed with the DirectShow run time and can be seen in the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, under the Video Capture Sources category. The VidCap sample is included so that you can view the source code for a video capture filter and see how to write capture filters.

Unlike ordinary DirectShow filters, special category filters, such as Video Capture Source filters, can work with more than one device.

Sample Locations

Transform Filters

This article details sample transform filters (sometimes called *effects*). The sample transform filters include an audio effect, some video effects, some MPEG codecs, a VCR controller, and various "null" (pass-through) filters.

Contents of this article:

- Contrast Sample (Video Contrast Filter)
- EzRGB24 Sample (Image Effect Filter)
- Gargle Sample (Gargle Filter)
- Inftee Sample (Infinite-Pin Tee Filter)
- MPGAudio Sample (MPEG Audio Decoder Filter)
- MPGVideo Sample (MPEG Video Decoder Filter)
- Nullip Sample (Null In Place Filter)
- Nullnull Sample (Minimal Null Filter)
- Vcrctrl Sample (VCR Control Filter)

Contrast Sample (Video Contrast Filter)

The video contrast sample filter, Contrast, illustrates how to define and implement a simple custom interface within the structure provided by the DirectShow™ base classes. This filter demonstrates how to use the CTransformFilter class to implement a simple effect filter.

The Contrast filter also provides a good example of the flexibility of the filter graph architecture. This flexibility is demonstrated when the filter is used in conjunction with a tee filter to produce "before and after" (side-by-side original and modified) video streams.

The Contrast filter is a simple transform filter that adjusts the contrast of the video stream that is passed through it. It provides a custom interface for adjusting the contrast. The Contrast filter also uses the CBasePropertyPage class to provide a property page for applications that do not provide a user interface.

Note The Contrast filter adjusts the contrast by using a trick with palettes. The color palette of an image effectively determines how the image is interpreted. By changing the palette, the filter can change the contrast without changing the image pixels themselves.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

EzRGB24 Sample (Image Effect Filter)

The RGB 24 image effect filter, EzRGB24, is a sample of an image processing filter. This shows usage of a number of DirectShow classes and interfaces, including CTransformFilter, CPersistStream, CBasePropertyPage, and ISpecifyPropertyPages.

This filter's purpose is to provide fast and single stream effects. The EzRGB24 filter also shows how to add image processing effects using DirectShow. You can use this filter as a component of an DirectShow video editor. Effects with fast execution were selected to help demonstrate the power of DirectShow in enabling real-time effects — the idea was to show something that could not have been built by using Microsoft Video for Windows®.

This filter performs a number of individual video effects. These include red, green, blue, darken, XOR, blur, gray, and emboss image-processing effects.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Gargle Sample (Gargle Filter)

The audio processing filter, Gargle, illustrates how to create a simple sound effect. More sophisticated effects such as echo, flanging, band-pass filtering, and other effects can be created in a way similar to that demonstrated by the Gargle filter. This sample demonstrates the CTransInPlaceFilter, CPersistStream, and CBasePropertyPage classes. It also shows use of the ASSERT, DbgBreak, DbgLog, and DbgBreakPoint debug macros.

The Gargle filter modulates the waveform passing through it by multiplying the waveform by another waveform that is mathematically generated within the filter. The modulating waveform is, by default, a triangular wave. The property sheet also offers the alternative of a square wave. You can set the frequency of the modulating wave through the filter's property sheet. At low frequencies (near 1 Hz), the sound grows and diminishes. At medium frequencies (5 to 15 Hz), the sound has a tremolo quality, which is why it is referred to as gargling. At higher frequencies (100 Hz and up), the filter generates extra frequencies in the original sound. If a 500 hertz (Hz) sound is played at 100 Hz, then it produces additional frequencies of 400 Hz, 600 Hz, 200 Hz, 800 Hz, and so on.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Inftee Sample (Infinite-Pin Tee Filter)

The Inftee sample infinite-pin tee filter has multiple output pins and passes through type enumerators from source filters. The filter uses the CAutoLock, CBaseFilter, CBaseInputPin, CBaseOutputPin, CCritSec, and COutputQueue base classes.

Inftee has one input pin and a variable number of output pins, typically two. It accepts data samples through the IMemInputPin transport, which it implements. All data samples sent to the filter are delivered down all paths simultaneously, therefore *teeing* the input into multiple separate output streams. The data samples are not type-specific, so the input, for example, can be text lyrics, video images, or audio buffers.

When considering two output pins, the tee filter sends the same data down both of the pins; therefore, the pins must have negotiated the same media type during connection. The infinite-pin tee filter handles this negotiation so that the input pin and both output pins converge when using the same media type. If a suitable media type cannot be found, then the connection is rejected.

The filter always uses the suggested allocator; the filter that provides the data suggests the allocator. The data arriving at the input pin is not copied before it is sent to the output pins. The filter also ensures that the data is delivered to the downstream filters, to guarantee that both outputs receive timely service. In particular, if one of the outputs can block in the COutputQueue::Receive member function, then the tee spins off a thread to deliver the sample. If there were no thread to deliver the sample, then the thread that delivers the sample to the tee input pin might pass the data to a downstream filter; at that point, it might block, keeping data from the other downstream filter for long periods of time.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

MPGAudio Sample (MPEG Audio Decoder Filter)

The sample MPEG audio codec, MPGAudio, provides a prototype for an MPEG audio decoder. It uses the CTransformFilter class.

As supplied, this framework just consumes the passed-in audio frames. That is, audio frames do not produce any output from this framework. You would need to expand the framework with code specific to the outputs desired.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

MPGVideo Sample (MPEG Video Decoder Filter)

The sample MPEG video codec, MPGVideo, provides a prototype for an MPEG video decoder. It uses the CTransformFilter class. The MPGVideo filter also shows the processing of quality-management messages.

As supplied, this framework just consumes the passed-in video frames. That is, video frames do not produce any output from this framework. You would need to expand the framework with code specific to the outputs desired.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Nullip Sample (Null In Place Filter)

The null in-place sample filter, Nullip, is an example of a transform-inplace filter. It illustrates how a transform filter can behave in a relatively transparent manner. Nullip is a simple filter that passes all data from its input pin to its output pin. No transformations are performed on the data.

The Nullip filter provides an example of using the CTransInPlaceInputPin class. It shows how to use the CAutoLock class to automatically release critical sections. Other classes used include CTransInPlaceFilter, CBasePropertyPage, CMediaType, and CTransInPlaceOutputPin. It also enables a user to select the media types that it can pass through itself. This filter has one input pin, one output pin, and performs its transform in place (without copying the data) in the push thread. In other words, the CTransInPlaceFilter::Receive method is called with a buffer, which it transforms and delivers to the next filter downstream. The Receive method is then blocked until that filter returns; it subsequently returns to its own calling member function.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Nullnull Sample (Minimal Null Filter)

The minimal null sample filter, Nullnull, illustrates a minimal filter. It does not support the media type selection that Nullip does. The Nullnull filter uses the CTransInPlaceFilter class.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Vcrctrl Sample (VCR Control Filter)

The VCR control sample filter, Vcrctrl, is a simple implementation of the external device control interfaces that DirectShow provides. Vcrctrl provides basic transport control and SMPTE timecode-reading capabilities for certain Betacam and SVHS videocassette recorders with RS-422 or RS-232 serial interfaces (see source code for specific machine types supported).

This sample is not intended to be a frame-accurate, system-ready implementation ready for professional applications. It is designed to show basic device control filter structure, and therefore does not implement several of the more sophisticated features of the interfaces, such as edit event control. Frame-accurate control is best achieved by writing a low-level, kernel-mode communications driver underneath the filter. Developers should refer to the Driver Development Kit (DDK) for the appropriate platform.

Using the Sample

With this sample, you can control a VCR through property pages. You should keep in mind the following:

- The sample automatically links the transport to the graph's Run, Pause, and Stop methods. You can unlink the transport on the General property page by removing the selection from the Link to Graph check box. This is a persistent property, so if you save the graph the transport will remain unlinked.
- The filter automatically detects the presence of a compatible VCR on COM2. If it can't find a VCR or if the port is in use, it switches to simulation mode. You can select the port to use on the General property page.
- The timecode pin can connect either to the text renderer or the timecode renderer (Tcrender.ax), which accepts the media type MEDIATYPE_AuxData/MEDIASUBTYPE_Timecode.
- The main property page shows the available video and audio inputs, although you can't select them. The hardware does not support this feature, and it was implemented to show physical pin property enumeration.
- The device communications class (CDevCom, specified in Cdevcom.h and Cdevcom.cpp) is thread-safe and handles SMPTE timecodes. On some VCRs, timecode detection is not supported, so timecode values might not be accurate.

This sample contains the following files:

- Vcrutil.h, .cpp — implementation of the device control interfaces
- Fvcrctrl.h, .cpp — the filter implementation
- Cdevcom.h, .cpp — the communications object that handles device-specific protocols and

the serial port interface
- Ctimecod.h, .cpp — the SMPTE timecode support
- Trprop.h, .cpp — the transport properties
- Vcrprop.h, .cpp — the general properties
- Vcrprop.rc — the dialog resources
- Vcrctrl.def — the module definition file
- Vcruids.h — the CLSIDs used by the filter

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Renderer Filters

This article describes sample renderer filters. The filters render video output, audio output, text output, and raw file output.

Contents of this article:

- Dump Sample (Dump Filter)
- SampVid Sample (Video Renderer Filter)
- Scope Sample (Oscilloscope Filter)
- TextOut Sample (Text Display Filter)

Dump Sample (Dump Filter)

The sample file dump filter, Dump, illustrates the use of the base filter class CBaseFilter and the rendered input pin class CRenderedInputPin. This sample also uses the IFileSinkFilter interface. The Dump filter demonstrates how to override the Receive method of the rendered input pin class to process actual media samples. The filter is also a "renderer" of its input stream. (That is, this filter doesn't fit the traditional definition of rendering audibly or visibly, although it is a "renderer filter.") The Dump filter delivers the EC_COMPLETE notification to the filter graph when it receives a call to **CDumpInputPin::EndOfStream** on its input pin.

This filter is a useful debugging tool. For example, you can verify, bit by bit, the results of a transform filter. You can build a graph manually by using the Filter Graph Editor, and connect the Dump filter to the output of a transform filter or any other output pin. You can also connect a tee filter and put the Dump filter on one leg of the tee filter and the typical output on another leg to monitor the results in a real-time scenario.

The Dump filter has a single input pin, which is dumped to a file. The filter prompts the user for a file name when it is instantiated and closes the file when it is freed.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

SampVid Sample (Video Renderer Filter)

The sample video renderer filter, SampVid, displays video inside a window. SampVid demonstrates the implementation of a special memory allocator, the CImageAllocator class, which uses buffers based on the Microsoft Win32® DIBSECTION structure. This sample shows use of the CBaseVideoRenderer class, which handles all the seeking, synchronization and quality management necessary for video renderers. Other base classes used include, among others:

- CBaseControlVideo
- CBaseControlWindow
- CDrawImage
- CImageAllocator
- CImageDisplay
- CImagePalette
- CMediaType

This sample also shows the use of the property page interface IQualProp, which addresses quality management.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Scope Sample (Oscilloscope Filter)

The oscilloscope sample filter, Scope, illustrates the use of the base filter class CBaseFilter and the base input pin class CBaseInputPin.

Other than the video renderer sample, Scope is the only sample filter that has a constantly active window.

Note The Scope window is implemented as a dialog box, not as an actual window.

Developers creating control panels to alter filter parameters in real time might want to use a technique like this rather than property pages.

The Scope filter demonstrates setting up a separate thread to process data. In this case, the data is just copied to a separate buffer on the IMemInputPin::Receive method, and is then drawn on the Scope window on the separate thread.

The filter demonstrates how to apply DirectShow™ outside the rather narrow domain of multimedia playback. DirectShow has wide application in the diverse areas of:

- Process control
- Factory automation
- Image processing and analysis
- Virtual lab instrumentation

All of these areas have semi real-time requirements, but additionally emphasize user interaction and the display of gauges, monitors, signal generators, alarms, and other forms of graphical display of analyzed results.

The Scope filter also enables you to monitor audio output to determine if you are clipping, so you can adjust the gain.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

TextOut Sample (Text Display Filter)

The sample text renderer filter, TextOut, serves as an example of a renderer filter that is for a user-defined type. It can render text data found in an AVI file or other source. The TextOut filter demonstrates the use of the CBaseRenderer class, which handles all the synchronization and seeking although it doesn't have any quality management implementation. To get quality management, derive your classes from CBaseVideoRenderer instead. This sample also uses the CBaseWindow class.

This renderer creates and uses a simple window to display each piece of text it receives.

This filter is installed with the DirectShow run time and is available through the Filter Graph Editor when you choose **Insert Filters** from the **Graph** menu, and select the DirectShow Filters category.

Sample Locations

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Cutlist Sample Applications

This article describes the cutlist sample applications included with the DirectShow SDK.

Contents of this article:

- CLText Sample (Text Cutlist Application)
- Simplecl Sample (Cutlist Application)

CLText Sample (Text Cutlist Application)

The text cutlist (CLText) sample reads a list of up to 150 cutlist elements (video or audio clips) from a text file and plays them.

This sample demonstrates how to use the ICutListGraphBuilder, IStandardCutList, and IFileClip interfaces.

Sample Locations

Simplecl Sample (Cutlist Application)

The simple cutlist (Simplecl) sample demonstrates how to use cutlists. Simplecl provides a File Open dialog box from which the user can choose a file to add to a cutlist. For each file, the user specifies a starting position and ending position for the clip. For every AVI file specified, the sample tries to add the first video stream and the first audio stream to its respective cutlist. The user must add at least two files, and then can run the filter graph and see the clips played sequentially.

This sample demonstrates how to use the ICutListGraphBuilder, IStandardCutList, and IFileClip interfaces.

Sample Locations

Previous Home Topic Contents Index Next

C/C++-based Sample Applications

This article details sample applications that are written in C++ or C. The sample applications implement media players or perform other application-related functions. The differences between them are mainly the different programming languages, API, or frameworks used.

Some of these samples use the Microsoft® Foundation Classes (MFC).

Contents of this article:

- AMCap Sample (DirectShow Capture Application)
- CPlay Sample (C/COM-based Media Player Application)
- Dvdsampl Sample (DVD Player Application)
- InWindow Sample (Window Playback Application)

- IPlay Sample (Indeo Player Application)
- MFCPlay Sample (C++/COM-based Media Player Application)
- MPEGProp Sample (MPEG Property Page Display Application)
- PlayFile Sample (Simple Playback Application)
- ShowStrm Sample (Multimedia Streaming Application)
- VidClip Sample (Video Editing Application)

AMCap Sample (DirectShow Capture Application)

The DirectShow capture sample, AMCap, demonstrates the basics of capturing audio and video from a hardware source to a specified output file or to a preview window.

To see a list of the hardware capture devices on your system, choose the **Devices** menu; DirectShow uses the ICreateDevEnum interface to create the list of devices that appears. After choosing a device, choose the **Set Frame Rate** command from the **Capture** menu and set the desired frame rate; the default value is 30 frames per second. To set an output file for your capture, pick the **Set Capture File** command from the **File** menu. DirectShow uses the File Writer filter to associate your specified file with the capture session.

If you want to turn audio capturing on or off, choose the **Capture Audio** command from the **Capture** menu. To turn capture previewing on or off, choose the Preview command from the Options menu. When you are ready to begin capturing, choose the **Start Capture** command from the **Capture** menu. If you want to save your captured data to a file other than the specified output file, choose the **Save Captured Video As** command from the **File** menu.

Note: Before you begin capturing for the first time, choose Allocate File Space from the File menu to preallocate the capture file. Preallocating the file improves capture performance. AMCap calls AllocCapFile to preallocate the capture file.

Sample Locations

CPlay Sample (C/COM-based Media Player Application)

This C/COM-based media player sample, CPlay, is a simple application that renders multimedia files using DirectShow™ from within the C language. The CPlay sample does not use Microsoft Foundation Classes (MFC). If you want to see a sample that uses MFC, see MFCPlay Sample (C++/COM-based Media Player Application). The CPlay sample shows how to utilize the DirectShow components without using the base classes provided with DirectShow. The sample creates a filter graph (by calling CoCreateInstance to get a pointer to the IGraphBuilder interface), and requests the filter graph to render a file. The filter graph is then controlled by using the IMediaControl interface. This sample also uses the IMediaEvent and IMediaPosition interfaces.

CPlay is a minimal application that implements the following menu commands: Open, Play, Pause, Stop, and Exit on the File menu, and About on the Help menu.

Sample Locations

Dvdsampl Sample (DVD Player Application)

The DirectShow DVD sample player, Dvdsampl, is a simple application that plays DVD Video content. It uses the IDvdGraphBuilder interface to build a DVD filter graph and to obtain interface pointers for IAMLine21Decoder, IDvdControl, and IDvdInfo. It uses those interfaces and enables you to turn closed captioning on and off, play in full screen mode, and display and select from DVD menus.

Note: Your system must include DVD playback hardware to run this sample.

Sample Locations

InWindow Sample (Window Playback Application)

The InWindow application is a simple sample that shows minimal code required to play back a media file into a specific window. It builds on the PlayFile sample. Like PlayFile, it provides a file Open dialog box that enables you to open media files. It also uses IGraphBuilder, IMediaEventEx, and IMediaControl as PlayFile does. InWindow expands on the PlayFile functionality by calling IVideoWindow::put_Owner, IVideoWindow::put_WindowStyle, and IVideoWindow::SetWindowPosition to direct playback into the main application window.

Sample Locations

IPlay Sample (Indeo Player Application)

The Intel® Indeo player sample, IPlay, uses DirectShow to render multimedia files and demonstrates the Indeo Video Interactive application programming interfaces (API). IPlay demonstrates how to access the advanced features of the Indeo Video Interactive (IVI) codec. It is a simple application written in C++ using Microsoft Foundation Classes (MFC). If the file is an IVI format file, it enables controls for the IVI advanced features. The advanced features include local decoding, a decoding time limit, and the ability to adjust brightness, saturation, and contrast.

The Ax_spec.h file defines the interfaces of the IVI codec. The CIPlayDoc::OnOpenDocument function demonstrates how to determine if a file is an IVI format file. The CIPlayDoc::Getxxx and CIPlayDoc::Setxxx functions demonstrate how to get and set the IVI playback parameters for the advanced features.

This sample uses the following interfaces:

- IGraphBuilder — to create a filter graph and render a file
- IFilterGraph — to enumerate the filters in the filter graph
- IMediaControl — to play/pause/stop the playback
- IMediaEvent — to be notified of EC_COMPLETE (playback complete) and other events
- IMediaPosition — to set the time that the media stream begins
- IVideoWindow — to set the window size/caption/position
- IBasicVideo — to get the native window size.

IPlay also implements and uses the **IIndeoDecode** interface (not part of DirectShow).

Sample Locations

MFCPlay Sample (C++/COM-based Media Player Application)

The C++/COM-based media player sample, MFCPlay, is a simple application that renders multimedia files using DirectShow from within C++ and Microsoft Foundation Classes (MFC). It demonstrates how to connect the DirectShow components by using the interfaces provided with DirectShow and MFC. The sample creates a filter graph (by calling CoCreateInstance to get a pointer to the IGraphBuilder interface), and requests the filter graph to render a file. The filter graph is then controlled by using the IMediaControl interface. This sample also uses the IMediaEvent and IMediaPosition interfaces.

The MFCPlay application implements the following menu commands: Open, Play, Pause, Stop, and Exit on the File menu, and About on the Help menu.

Sample Locations

MPEGProp Sample (MPEG Property Page Display Application)

The MPEGProp sample application demonstrates how to display a filter's property page.

When you open MPEGProp, choose Open from the File menu and select an MPEG media file from the standard Open dialog box. After you select an appropriate file, MPEGProp calls CoCreateInstance to create a new filter graph and renders the chosen source file. MPEGProp calls IFilterGraph::FindFilterByName to locate the MPEG Video Codec filter, and then calls OleCreatePropertyFrame with the codec pointer to display the filter's property page.

Sample Locations

PlayFile Sample (Simple Playback Application)

The PlayFile application is a simple sample that shows minimal code required to play back a media file. It provides a file Open dialog box that enables you to open files including file types such as AVI, MPEG, MOV, and QT. PlayFile uses the IGraphBuilder::RenderFile method to render the filter graph for the chosen media file, IMediaEventEx to handle signaling of events, IMediaControl::Run to play the resulting filter graph, and IMediaControl::Stop to stop playback. The sample calls IVideoWindow to control whether the playback window is visible. Video follows the default behavior and plays back in a separate window rather than in the main application window.

See InWindow Sample (Window Playback Application) for a sample that builds on PlayFile and plays back video into the main application window.

Sample Locations

ShowStrm Sample (Multimedia Streaming Application)

The ShowStrm application demonstrates how to use a Microsoft® DirectDraw® surface to blit a multimedia stream. It is a console application that sends a movie out to the surface when invoked from the command line with the following syntax:

```
SHOWSTRM Name_of_Movie
```

For example, to play Angry.avi, you could use the command:

```
C:\>SHOWSTRM angry.avi
```

Sample Locations

VidClip Sample (Video Editing Application)

The VidClip application demonstrates how to use the multimedia streaming interfaces and how to support rudimentary video editing. The VidClip sample reads from multiple streams and writes to a single stream. For more information on the multimedia streaming interfaces, see List of Multimedia Streaming Interfaces.

On the **Video** menu in the VidClip application, choose the **Add Clip** command to add clips to your list; you can include start and stop times, if you want to add only a portion of a clip. Choose the **Edit Clip** command to edit existing clips in the list, and the **Delete Clip** command to delete clips. The **Make a Movie** command combines the clips into one stream.

On the **File** menu, choose the **Settings** command to set the height, width, depth, and compression for the stream data. Choose the **Save** command to save the settings to a file in your project.

Sample Locations

Visual Basic-based Sample Applications

This article describes the Microsoft® Visual Basic® version 5.0 samples in the Microsoft DirectShow™ Software Development Kit (SDK).

Contents of this article:

- Visual Basic-Based ActiveX Player
- Visual Basic-Based Filter Graph Builder
- Visual Basic-Based Filter Graph Player
- Visual Basic-Based Player

Visual Basic-Based ActiveX Player

This ActiveX sample, Ocxvb01, demonstrates an application that uses the ActiveMovie Control. It has a form that contains menus and controls to demonstrate the access of various properties of the video control. See Using the ActiveMovie Control in Visual Basic for a detailed description of this sample.

The sample has a menu command that enables you to create the control by opening a video file, and to use Run, Pause, and Stop commands to programmatically control the ActiveMovie Control. All of the ActiveMovie Control Panel features, such as selection controls, position controls, and the trackbar control, can be made visible selectively, and enabled or disabled from menu commands in the sample application. Other properties, such as the rate and current position, can also be changed from the application, and you can set the image in the video control to various sizes such as half, default, double, and full-screen.

Sample Locations

Visual Basic-Based Filter Graph Builder

The filter graph builder sample, Builder, demonstrates how a Visual Basic-based application uses the collection interfaces that the filter graph manager exposes to build and connect a custom filter graph. The collection interfaces are composed of the following interfaces.

- IAMCollection
- IRegFilterInfo
- IFilterInfo
- IPinInfo
- IMediaTypeInfo

These interfaces appear as objects in the ActiveMovie Control type library in the Visual Basic development environment and enable the registry to be searched, filters to be added, and pins to be connected, among other things. See Constructing Filter Graphs Using Visual Basic for a detailed description of this sample.

The Builder sample presents a form that enables the user to see a list of filters in the registry and add filters to the filter graph. It also enables the user to examine the names and properties of pins on the filter, the media types on the pins, and to connect the pins. The sample has an additional sample routine, accessed by a menu, that automatically builds a filter graph. The filter graph is built using the collection interface objects, to better demonstrate the more typical use of these objects in a Visual Basic-based application. The sample also enables you run the constructed filter graph.

Sample Locations

Visual Basic-Based Filter Graph Player

The filter graph player, VBDemo, demonstrates how to use the DirectShow Visual Basic objects that control the playback of video and that adjust properties of various components. See Controlling Filter Graphs Using Visual Basic for a detailed description of this sample.

The VBDemo sample enables you to open a video file. Use the Play, Pause, and Stop buttons to control the video. It positions the video window as a child window of the main form and displays the movie's length and elapsed time. It also enables you to set the start position and rate. Use two sliders to adjust the audio renderer's volume and balance.

Sample Locations

Visual Basic-Based Player

The sample player, Player, is a short Visual Basic-based program that demonstrates the minimum requirements for playing back a DirectShow media file without using the ActiveMovie Control. It uses the same DirectShow object library as the VBDemo sample, but provides only three buttons (Open, Play, and Stop) and a slider for indicating position.

Sample Locations

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

PID Sample (Plug-In Distributor Application)

The IAMovie sample shows the implementation of the IAMovie interface for a plug-in distributor (PID). A PID is used to extend the filter graph manager. The filter graph manager distributes the actions of interface methods to the appropriate filters. This distribution enables applications to have a single point of control to perform the basic operations.

In the IAMovie sample, the PID exposes the IAMovie interface and implements it by calling the enumerator of the filter graph manager, finding which filters expose the interface (see EnumFiltersByInterface), and communicating directly with those filters. PIDs are supplied for the standard control interfaces. Independent software vendors (ISVs) can replace these supplied PIDs and add others.

See Plug-in Distributors for information on plug-in distributors.

Sample Locations

SampIOS Sample (IOStream Helper Library)

The IOStream helper library provides text output of the IBaseFilter interface and other DirectShow™ objects. This sample code is provided as a header file (SampIOS.h) and a source file (SampIOS.cpp) that can be built into a library (SampIOS.lib). You can use the library functions in your DirectShow filters and applications. The functions are designed to help retrieve information in text format about the objects in the filter graph. The text output is intended to aid in debugging.

SampIOS.h declares debug output functions that you can call from C. For example, DumpFilterInfo sends information on the given filter interface. See the header file (SampIOS.h) for the syntax of each call. The DumpXxx functions require the cout output stream to be valid in the environment in which you are working, as they implicitly output to **cout**. The DumpXxx functions are included in the following list.

- DumpFilterInfo
- DumpPinInfo
- DumpAllPins
- DumpAllFilters
- DumpFilterGraph

Extensions to the C++ IOStream library are included, so you can write code like the following to dump information to cout.

```
    cout << piFilterGraph << endl;
```

The C++ callable functions declared by SampIOS.h include the following abilities to provide output to an ostream. (The standard C++ ostream class provides the basic capability for sequential and random-access output.) See the header file (SampIOS.h) for the syntax of each call.

- Provide output of a wide string.
- Build and provide output of a textual version of a GUID.
- Provide output of filter information.
- Provide output of filter information for all filters left in an enumerator.
- Provide output of filter and connection details.
- Provide output of pin details (including connection information).
- Provide output of pin details (including connection information) for all pins left in the enumerator.
- Provide output of the pin information for all pins on the supplied filter.

Sample Locations

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Build Information

To use some of the filters and applications supplied as DirectShow samples, you need to build them into executable or DLL (.ax) files. To build, you need various tools. And after you build a filter, you need to register it with a unique identifier to make it appear in the Filter Graph Editor. This section contains the following topics:

- Tools for Building the Samples
- Building the Samples
- Setting Up the Registry

Tools for Building the Samples

The tools you need to build the filters and applications supplied as DirectShow samples are included in the following list. This list contains both required and optional tools.

- Microsoft® DirectShow™ Software Development Kit (required).
- ANSI-compatible C/C++ compiler, such as Microsoft Visual C++® version 5.*x* or later (required).

- Microsoft Win32® headers and libraries (required).
- Microsoft NMAKE (supplied with Visual C++), if you want to use the supplied makefiles (optional).
- Microsoft Foundation Classes (MFC), if you want to build the two sample applications that use MFC (optional).

Building the Samples

All samples in the SDK have been provided with a makefile that is compatible with NMAKE. In some cases, a Visual C++ project makefile is also included.

To build the samples using the standard makefiles, you must first set up your build tools (for example, Microsoft Visual C++), the Win32 headers and libraries, and the DirectShow SDK. The DirectShow SDK makefiles expect to locate the standard Windows® and Win32 include files and libraries on the INCLUDE and LIB paths. The makefiles handle all DirectShow-specific files on a project-by-project basis.

The makefiles provided for each individual sample use ActiveX.mak, which is provided with the DirectShow SDK. If you are using ActiveX.mak to build for any machine, other than the default x86, you must set the CPU environment variable to indicate which kind of machine. The allowable values for the CPU are i386, ALPHA, and PPC.

Performance Note If you copy any sample makefile to create a new dynamic-link library (DLL) (including filters and PIDs) make sure that you change the base address to avoid collisions with other DLLs. A collision of DLL load address results in one of the DLLs having to be relocated during load time, thereby increasing the duration of load time. In the sample makefiles, the base address is set in DLL_BASE, which is used in ActiveX.mak. Do not let ActiveX.mak use the default value for DLL_BASE, as this will almost certainly cause collisions.

A makefile that builds all the samples is provided in the Samples directory.

Setting Up the Registry

The source directory for each sample contains a registry file (.reg). Sample filters are registered by default. You can re-register all filters in the sample by opening this file. The samples also use the preferred self-registration capabilities provided by the IAMovieSetup interface. See Register DirectShow Objects for more information.

Caution: If you modify a registry file and then use it to update your registry, you will affect the sample filter that is already installed on your system. For example, if you modify Ball.reg, you might lose access to the Ball filter (Ball.ax) in the Filter Graph Editor.

Multimedia Streaming

This section describes the multimedia streaming interfaces, which automatically negotiate the transfer and conversion of data from a source to an application. It includes reference entries for the multimedia streaming data types, interfaces, and interface methods. It also includes sample code and a description of multimedia streaming component objects along with their CLSIDs and the interfaces they support.

▪About the Multimedia Streaming Architecture

▪List of Multimedia Streaming Interfaces

▪Multimedia Streaming Reference

▪Multimedia Streaming Component Objects

▪Multimedia Streaming Sample Code

About the Multimedia Streaming Architecture

This article describes the architecture of multimedia streaming and how software developers typically use streams in their tools and applications. It also covers the advantages of streaming as a base for data transfer and how to address multimedia programming issues, such as data transfer, performance optimization, and time stamping in streaming applications. Programmers who want to use multimedia streaming should be familiar with COM programming concepts.

Contents of this article:

- Advantages of Multimedia Streaming
- Object Hierarchy
- Creating Multimedia Stream Objects and Stream Samples
- Using Multimedia Streams in Applications
- Sharing Data Between Streams

Advantages of Multimedia Streaming

When developers use multimedia streaming in their applications, it greatly reduces the amount of format-specific programming needed. Typically, an application that must obtain media data from a file or hardware source must know everything about the data format and the hardware device. The application must handle the connection, transfer of data, any necessary data conversion, and the actual data rendering or file storage. Because each format and device is slightly different, this process is often complex and cumbersome. Multimedia streaming, on the other hand, automatically negotiates the transfer and conversion of data from the source to the application. The streaming interfaces provide a uniform and predictable method of data access and control, which makes it easy for an application to play back the data, regardless of its original source or format.

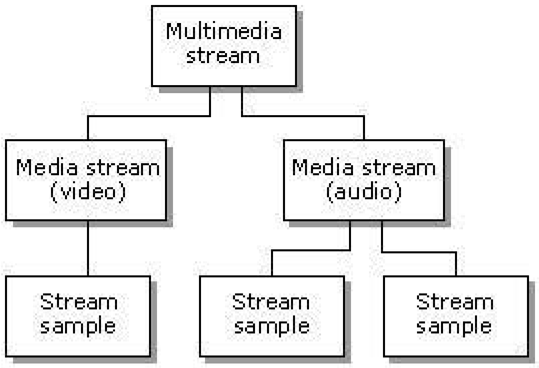
The following steps show how to implement streaming, from hardware device to rendered playback.

1. A source of video data, such as Microsoft® DirectShow™, exposes the streaming interfaces.
2. The application developer uses the multimedia streaming interfaces to handle data format conversion.
3. The application developer uses the Microsoft DirectDraw® interfaces to render the resulting data.

The specification for multimedia streams comprises several interfaces; each interface includes methods that control a certain aspect of the streaming process or handle a certain type of data. See List of Multimedia Streaming Interfaces for additional information.

Object Hierarchy

The following diagram shows the basic object hierarchy used in multimedia streaming.



There are three basic object types defined in the multimedia streaming architecture:

1. A multimedia stream, which supports the IMultiMediaStream interface.
2. Media streams support the IMediaStream interface and are data specific. Every multimedia stream contains one or more of these media streams.
3. Stream samples support the IStreamSample interface and are created by a media stream. These objects represent a basic unit of work for the stream.

Creating Multimedia Stream Objects and Stream Samples

Objects that support the IMultiMediaStream interface are the basic containers for multimedia data streams. The **IMultiMediaStream** interface includes methods that enumerate the object's data streams; these streams are typically video and audio data, but can include data of any format, such as closed-captioning, plain text, or SMTPE timecode. The

IMultiMediaStream interface is a generic container, however; developers can create other versions of the interface that support specific data formats. Objects that implement the IAMMultiMediaStream interface, for example, can enumerate and control streams of any DirectShow data format. Because individual data streams are format specific, they support at least two different interfaces: one generic and one data-specific. Every stream supports the IMediaStream interface, which provides methods to retrieve its format and a pointer to the stream itself. The IDirectDrawMediaStream interface, on the other hand, has methods that deal specifically with rendering video data. Any interface derived from **IMultiMediaStream** also supports the creation of stream samples, the basic units of streaming data.

A multimedia sample is a reference to an object containing the media data. For a video image, this is a DirectDraw surface. The sample's exact content varies, depending on the type of media (sound, text, and so on). Because a sample is only a reference to the data object, any number of stream samples can refer to the same object. The IStreamSample interface provides methods that get and set a sample's characteristics, such as its start and stop time, status, and stream association. The IStreamSample::Update method refreshes the sample's data in the case of readable streams. For writable streams, it will write the sample's data to the stream. Typically, you use the Update method in a loop that renders, transfers, or stores streaming data. See Use Multimedia Streaming in DirectShow Applications for a practical example of this method in source code.

Using Multimedia Streams in Applications

The multimedia streaming interfaces greatly simplify the process of manipulating multimedia data by removing the dependency on specific characteristics of the hardware or software source and providing support for all Microsoft DirectX® media formats. Streams abstract the data to a very high level; applications can even move data from one stream to another without knowing anything about the data's format.

Perform the following steps to create a multimedia stream:

1. Create the multimedia stream. The method of creation and initialization of the stream is architecture specific. DirectShow supports the IAMMultiMediaStream interface, which is used to initialize the stream. Other in-process server implementations of IMultiMediaStream will be created and initialized using different mechanisms.
2. After the multimedia stream object is initialized, the application will use QueryInterface to retrieve the IMultiMediaStream interface for the object. Use this interface to determine the stream's properties and enumerate the streams themselves. You can retrieve a specific stream by calling the IMultiMediaStream::GetMediaStream method with a specific purpose ID. MSPID_PrimaryVideo and MSPID_PrimaryAudio, which represent the primary video and audio streams, are the most commonly used purpose IDs.
3. Call IUnknown::QueryInterface for an interface specific to the stream's media type. If you want to render a video stream, for example, retrieve its IDirectDrawMediaStream interface. Media-specific interfaces define additional methods necessary for taking full advantage of a format's capabilities.
4. Create one or more samples from the stream data. Every media stream supports the IMediaStream::CreateSharedSample method for sample creation. The resulting sample supports the IStreamSample interface, which provides control over the sample and its characteristics. Typically, the media stream supports a format-specific method of sample creation that is more powerful than the aforementioned **IStreamSample** methods. IDirectDrawMediaStream, for example, can create samples attached to a desired DirectDraw surface and clipping rectangle. In some situations, however, you must handle data without knowing about its data format, such as when constructing a cutlist. If you want to stream data independent of its format, use the **IMediaStream::CreateSharedSample** method to create the data samples.

5. After creating all desired stream samples, start the stream by calling the IMultiMediaStream::SetState method and pass in the STREAMSTATE_RUN flag as its parameter.
6. Call IStreamSample::Update to update the stream sample. When the **IStreamSample::Update** method exits, you can access the sample's data. The code sample in Use Multimedia Streaming in DirectShow Applications uses the **IStreamSample::Update** method in a loop to render a stream of video data. If you want a trigger a specific event or function call when the update returns, pass the appropriate pointers to the **IStreamSample::Update** method.

For more information on the multimedia streaming interfaces, see .

Sharing Data Between Streams

Processing multimedia data typically requires a great deal of system resources; therefore, you should avoid copying data whenever possible. The streaming architecture supports shared stream samples, a mechanism that moves data from one stream to another without copying it. This buffer enables the efficient transportation of data between two streams even if the destination stream doesn't specifically support the underlying data format.

For example, assume that you have a multimedia stream with three data streams: video and audio, and URL data time-stamped to match the video content. You want to write an application that adds a copyright notice on every video frame and writes the data to another stream for storage, but your application doesn't understand any data formats except the video stream. For the video stream, you create a sample attached to the desired DirectDraw surface. You can then create an output stream by calling either the IDirectDrawMediaStream::CreateSample method with that pointer to the same surface, or IMediaStream::CreateSharedSample. In both cases, the input and output streams share the DirectDraw surface. Because you understand the video format, you can access this surface as needed.

To retrieve the other source stream pointers (audio and URL), enumerate the source container stream and grab pointers to the nonvideo streams. Each of these source streams has an associated output stream in the output stream container. Retrieve these output pointers by calling the IMultiMediaStream::GetMediaStream method on the output container with each of the source stream pointers. The following steps describe this process.

1. Call IMultiMediaStream::EnumMediaStreams to retrieve a pointer to a source stream. Make sure that it's not the video stream, because your application already understands its format.
2. Call IMultiMediaStream::GetMediaStream on the output container stream with the pointer from step 1. This returns a pointer to the desired output stream.
3. Call AllocateSample on the source stream.
4. Call CreateSharedSample on the output stream.
5. Call Update on the source stream to read the data.
6. Call Update on the output stream to write the data.

Repeat these steps for each stream whose format you don't support. When both samples finish updating, the output stream has all data from the source stream and you are done.

List of Multimedia Streaming Interfaces

Multimedia streaming comprises a number of interfaces; this article provides a brief description of each interface. For additional information about multimedia streaming, see About the Multimedia Streaming Architecture.

Contents of this article:

- Base Multimedia Streaming Interfaces
- DirectDraw Streaming Interfaces
- Audio Streaming Interfaces
- DirectShow Multimedia Streaming Interfaces

Base Multimedia Streaming Interfaces

The DirectX Media SDK includes reference documentation for the multimedia streaming interfaces; the interfaces and their methods provide a programatic way of access multimedia streams. However using a base interface to access a specific type of data can limit the amount of control you have over the data, so media developers should create derived versions of these interfaces that provide more powerful control over the unique capabilities of their media type.

IMultiMediaStream
: This interface defines how to access the highest-level multimedia stream object; this object contains and provides access to other stream objects. IMultiMediaStream has methods that enumerate or retrieve specific streams, as well as checking the stream's total time duration and seeking within the stream.

IMediaStream
: This interface defines a generic stream object. Use its methods to retrieve a pointer to the stream, get information about the stream, and create samples from the stream data. You can also create shared stream samples, which multiple streams can access without duplicating the sample's data.

IStreamSample
: This interface controls the behavior of a specific stream sample. You can retrieve the stream that created the sample, check the sample's start and end times and completion status, and perform a user-defined function on the sample itself (through the Update method). Typically, the **Update** method processes the sample data in an appropriate manner, such as rendering video data or playing back audio data.

DirectDraw Streaming Interfaces

If you use Microsoft® DirectDraw-supported video formats in your streams, the following interfaces give you more powerful control over the data than the more generic base interfaces.

IDirectDrawMediaStream
: This interface derives from IMediaStream; it sets and retrieves the stream format and the DirectDraw® object associated with the media stream. You can also use this

interface to create video samples.

IDirectDrawStreamSample
: This interface derives from the IStreamSample interface and enables you to attach video samples to DirectDraw surfaces. Each attached surface includes a clipping rectangle to make rendering easier.

Audio Streaming Interfaces

IAudioMediaStream
: The IAudioMediaStream interface controls audio media streams. This interface inherits from the IMediaStream interface and is used to create one or more IAudioStreamSample objects. It is also used to set and retrieve the current format of the stream data.

IAudioStreamSample
: The IAudioStreamSample interface retrieves information from the underlying IAudioData data objects.

IMemoryData
: The IMemoryData interface contains methods which set and retrieve memory data on audio data objects. Audio data objects provide the underlying data which stream samples access. This interface provides a way to initialize memory buffers and to set actual amounts of audio data in the objects. Additionally, the IMemoryData::GetInfo method can be used to retrieve audio memory data.

IAudioData
: The IAudioData interface provides methods which allow applications to set and get the underlying audio data that audio streams will reference. The audio data format is set in the WAVEFORMATEX structure.

DirectShow Multimedia Streaming Interfaces

The following interfaces expose Microsoft DirectShow™ functionality to multimedia stream developers. Application developers should implement only the IAMMultiMediaStream interface in their applications; DirectShow uses the other two interfaces internally.

IAMMultiMediaStream
: This interface controls connections between streams and DirectShow filter graphs. It provides methods that instantiate filter graphs, open DirectShow-supported media files, and generate an appropriate filter graph for playback. The CLSID_ActiveMovieMMStream class identifier supports this interface.

IAMMediaStream
: This interface is a DirectShow-specific derivation of the IMediaStream interface. It handles the internal negotiation of connections between multimedia streams and DirectShow filter graphs. Application developers should not use or implement this interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Multimedia Streaming Reference

This section contains reference entries for all the multimedia streaming interfaces and their methods, including those that DirectShow supports. It also includes a list of multimedia streaming data types that the interfaces use, and a list of error and success codes that the methods return.

- IMultiMediaStream Interface
- IMediaStream Interface
- IStreamSample Interface
- IDirectDrawMediaStream Interface
- IDirectDrawStreamSample Interface
- IAudioMediaStream Interface
- IAudioStreamSample Interface
- IMemoryData Interface
- IAudioData Interface
- IAMMultiMediaStream Interface
- IAMMediaStream Interface
- Multimedia Streaming Data Types
- Error and Success Codes for Multimedia Streaming

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMultiMediaStream Interface

The **IMultiMediaStream** interface provides methods that control a multimedia stream and provide access to its underlying media streams. A multimedia stream is the highest-level streaming object and can contain one or more media streams. While each media stream is media-type specific (audio, video, and so on), multimedia streams are generic across all types because they must provide access to a number of streams that can have different media types. **IMultiMediaStream** interface methods enable you to enumerate and retrieve pointers to the specific streams; IMediaStream interface methods provide specific control over the media stream behavior.

For sample code which implements the multimedia streaming interfaces see Multimedia Streaming Sample Code.

When to Implement

Implement this interface when you want create containers for a specific type of media stream.

When to Use

Use this interface when your application must enumerate and control a multimedia stream's underlying, type-specific streams.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IMultiMediaStream methods	Description
GetInformation	Retrieves the capabilities and stream type of a multimedia stream.
GetMediaStream	Retrieves a media stream that has the specified purpose ID.
EnumMediaStreams	Retrieves a media stream from a multimedia stream by zero-based index.
GetState	Retrieves the multimedia stream's current state.
SetState	Sets the media stream to either a running or stopped state.
GetTime	Retrieves the current time from the multimedia stream's clock, if it has a clock.
GetDuration	Retrieves the media stream's duration.
Seek	Sets the seek location of all derived media streams to the specified time.
GetEndOfStreamEventHandle	Retrieves the handle for the event triggered when the stream completes playback.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMultiMediaStream::EnumMediaStreams

IMultiMediaStream Interface

Retrieves a media stream from a multimedia stream by zero-based index.

HRESULT EnumMediaStreams(
  long *Index*,
  IMediaStream** *ppMediaStream* **)**

Parameters

Index
  [in] Index of the stream array to check.
ppMediaStream
  [out] Address of a pointer to an IMediaStream interface object. On return, it contains a pointer to the stream at the specified index.

Return Values

Returns one of the following values.

Value	Meaning
E_POINTER	The *ppMediaStream* pointer is invalid.
S_FALSE	*Index* is out of range; no streams are left to enumerate. When the method returns this value, it also sets *ppMediaStream* to NULL.
S_OK	Success.

Remarks

You should call this method until it returns S_FALSE, which indicates that the stream enumeration is complete.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMultiMediaStream::GetDuration

IMultiMediaStream Interface

Returns the media stream's duration.

HRESULT GetDuration(
  STREAM_TIME **pDuration*
)

Parameters

pDuration
[out] Pointer to a STREAM_TIME value that will contain the media duration.

Return Values

Returns one of the following values.

Value	**Meaning**
E_POINTER	The value of *pDuration* is invalid.
MS_E_WRITESTREAM	The media stream is writable and therefore has no duration.
S_FALSE	Stream contains live data or this method couldn't determine the duration. On return, this method sets *pDuration* to zero.
S_OK	Stream contains recorded media. On return, *pDuration* contains duration of media.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMultiMediaStream::GetEndOfStreamEventHandle

IMultiMediaStream Interface

Retrieves the handle for the event triggered when the stream completes playback.

HRESULT GetEndOfStreamEventHandle(
 HANDLE* *phEOS*
)

Parameters

phEOS
[out] Pointer to an event HANDLE returned by the current object when it completes playback. If no HANDLE is associated with the object, this value will be NULL.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Remarks

The Microsoft® Win32® WaitForSingleObject and WaitForMultipleObjects functions use the retrieved handle to watch for completion of playback.

IMultiMediaStream::GetInformation

IMultiMediaStream Interface

Retrieves the capabilities of a media stream that matches the specified media type.

HRESULT GetInformation(
  DWORD **pdwFlags*,
  STREAM_TYPE **pStreamType*
  );

Parameters

pwdFlags
[out] Pointer to a value that will contain a combination of one or more of the following flags. Can be NULL.

Value	**Meaning**
MMSSF_ASYNCHRONOUS	The stream supports asynchronous sample updates. All implementations of IMultiMediaStream will support the asynchronous updates; this flag confirms it.
MMSSF_HASCLOCK	The stream has a clock.
MMSSF_SUPPORTSEEK	The stream supports seeking.

pStreamType
[out] Pointer to a STREAM_TYPE enumeration type that will contain the media type information for the derived media streams. Can be NULL.

Return Values

Returns S_OK if successful.

Remarks

A stream's capabilities include whether it has a clock, if it supports seeking, and whether it supports asynchronous updating.

Previous Home Topic Contents Index Next

IMultiMediaStream::GetMediaStream

IMultiMediaStream Interface

Retrieves a media stream that has the specified purpose ID.

HRESULT GetMediaStream(
  REFMSPID *idPurpose*,
  IMediaStream ***ppMediaStream*
  );

Parameters

idPurpose
    Value that specifies the desired stream.
ppMediaStream
    Address of a pointer to an IMediaStream interface that will point to the desired media stream.

Return Values

Returns one of the following values.

Value	Meaning
E_POINTER	The *ppMediaStream* pointer is invalid.
MS_E_NOSTREAM	No stream has the specified purpose ID.
S_OK	Success.

Remarks

If a stream exists that matches the purpose ID in *idPurpose*, the *ppMediaStream* parameter points to the stream and increments its reference count.

MSPID_PrimaryVideo and MSPID_PrimaryAudio, which represent the primary video and audio streams, are the most commonly used purpose IDs.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMultiMediaStream::GetState

IMultiMediaStream Interface

Retrieves the multimedia stream's current state.

HRESULT GetState(
  STREAM_STATE* *pCurrentState*
)

Parameters

pCurrentState
: [out] Pointer to the STREAM_STATE enumerated type that will contain the current multimedia stream's state.

Return Values

Returns S_OK if successful or E_POINTER if *pCurrentState* is invalid.

Previous Home Topic Contents Index Next

IMultiMediaStream::GetTime

IMultiMediaStream Interface

Retrieves the current time from the multimedia stream's clock, if it has a clock.

HRESULT GetTime(
  STREAM_TIME* *pCurrentTime*
)

Parameters

pCurrentTime
: [out] Pointer to a STREAM_TIME value that will contain the current time, if the media stream has a clock.

Return Values

Returns one of the following values.

Value	**Meaning**
E_POINTER	The *pCurrentTime* pointer is invalid.
S_FALSE	Stream doesn't have a clock; **pCurrentTime* is zero.
S_OK	Stream has a clock and the method succeeded; *pCurrentTime* contains the current time.

Remarks

If the stream doesn't have a clock, this method sets **pCurrentTime* to zero and returns S_FALSE. If a stream has a clock, the stream sample times are relative to the stream's clock.

STREAM_TIME is defined as a **LONGLONG** value.

IMultiMediaStream::Seek

IMultiMediaStream Interface

Sets the seek location of all contained media streams to the specified time.

HRESULT Seek(
 STREAM_TIME *SeekTime*
)

Parameters

SeekTime
 [in] STREAM_TIME value that specifies the seek time.

Return Values

Returns one of the following values.

Value	**Meaning**
E_POINTER	One of the pointers is invalid.
MS_E_NOSEEKING	One or more media streams don't support seeking.
MS_E_WRITESTREAM	The streams are writable and therefore don't support seeking.
S_OK	Success.

Remarks

This method won't work on streams that don't support seeking. Before calling this method, retrieve the stream's capabilities by calling IMultiMediaStream::GetInformation; if the retrieved value includes the MMSSF_SUPPORTSEEK flag, you can call this method.

When seeking a stream that has a clock, the current time can change to an unpredictable value, including a time before the desired seek time. This causes the method to fail.

This method seeks to the specified time in all the media streams derived from the multimedia stream object.

Previous Home Topic Contents Index Next

IMultiMediaStream::SetState

IMultiMediaStream Interface

Sets the media stream to either a running or stopped state.

HRESULT SetState(
  STREAM_STATE *NewState*
)

Parameters

NewState
    [in] A STREAM_STATE enumeration value that specifies the new media stream state.

Return Values

Returns S_OK.

Remarks

When you set the stream to STREAMSTATE_STOP, this method deletes all data still pending.

IMediaStream Interface

The **IMediaStream** interface provides access to the characteristics of a media stream, such as the stream's media type and purpose ID. It also has methods that create data samples.

For sample code that implements the multimedia streaming interfaces, see Multimedia Streaming Sample Code.

When to Implement

Implement this interface when you want to add media type-specific functionality to your media stream. This interface is implemented on multimedia stream objects. IMediaStream provides generic sample-creation methods, but you usually want to write a more powerful version of these methods that will take advantage of your media type's specific characteristics.

When to Use

Use this interface when your application needs to access a stream's media type information and create data samples.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IMediaStream methods	**Description**
GetMultiMediaStream	Retrieves a pointer to the multimedia stream that contains the specified media stream.
GetInformation	Retrieves the stream's purpose ID and media type.
SetSameFormat	Sets the media stream the same format as a previous stream.
AllocateSample	Allocates a new stream sample object for the current media stream.
CreateSharedSample	Creates a new stream sample that shares the same backing object as the existing sample.
SendEndOfStream	Forces the current stream to end. If the current stream isn't writable, this method does nothing.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaStream::AllocateSample

IMediaStream Interface

Allocates a new stream sample object for the current media stream.

HRESULT AllocateSample(
  DWORD *dwFlags*,
  IStreamSample ***ppSample*
)

Parameters

dwFlags
  [in] Flags. Must be zero.
ppSample
  [out] Pointer to the newly created stream sample's IStreamSample interface.

Return Values

Returns one of the following values.

Value	**Meaning**
E_OUTOFMEMORY	There isn't enough memory available to create a stream sample.
E_POINTER	A parameter is invalid.
S_OK	Success.

Remarks

This method allocates the sample and its associated backing object or buffer. The backing object is either the DirectDraw surface for video or the IAudioData object for audio.

IMediaStream::CreateSharedSample

IMediaStream Interface

Creates a new stream sample that shares the same backing object as the existing sample.

HRESULT CreateSharedSample(
  IStreamSample* *pExistingSample*,
  DWORD *dwFlags* **,**
  IStreamSample** *ppNewSample*
)

Parameters

pExistingSample
  [in] Pointer to the existing sample.
dwFlags
  [in] Reserved for flag data. Must be zero.
ppNewSample
  [out] Address of a pointer to an IStreamSample interface that will point to the newly

created shared sample.

Return Values

Returns one of the following values.

Value	Meaning
E_OUTOFMEMORY	There isn't enough memory available to create the sample.
E_POINTER	One of the parameters is invalid.
MS_E_INCOMPATIBLE	The existing sample isn't compatible with the specified media stream.
S_OK	Success; *ppNewSample* points to the newly created sample.

Remarks

This method calls IUnknown::QueryInterface on the existing sample to retrieve the media type-specific information, which it uses to create the shared sample.

IMediaStream::GetInformation

IMediaStream Interface

Retrieves the stream's purpose ID and media type.

HRESULT GetInformation(
  MSPID* *pPurposeId*,
  STREAM_TYPE* *pType*
)

Parameters

pPurposeId
 [out] Pointer to an MSPID value that will contain the stream's purpose ID. If this parameter is NULL on entry, the method won't retrieve the purpose ID.

pType
 [out] Pointer to a STREAM_TYPE enumerated data type value that will contain the stream's media type. If this parameter is NULL on entry, the method won't retrieve the media type.

Return Values

Returns S_OK if successful or E_POINTER if one of the parameters is invalid.

Remarks

The value retrieved in the *pPurposeId* parameter will usually be either MSPID_PrimaryVideo, which identifies the primary video stream, or MSPID_PrimaryAudio, which identifies the primary audio stream; however, you can define other values if necessary.

IMediaStream::GetMultiMediaStream

IMediaStream Interface

Retrieves a pointer to the multimedia stream that contains the specified media stream.

HRESULT GetMultiMediaStream(
IMultiMediaStream** *ppMultiMediaStream*
)

Parameters

ppMultiMediaStream
[out] Address of a pointer to an IMultiMediaStream interface object that will point to the multimedia stream from which the current media stream was created.

Return Values

Returns S_OK if successful or E_POINTER if *ppMultiMediaStream* is invalid.

Remarks

This method increments the reference count of the retrieved object pointer.

IMediaStream::SendEndOfStream

IMediaStream Interface

Forces the current stream to end. If the current stream isn't writable, this method does nothing.

HRESULT SendEndOfStream(
  DWORD *dwFlags*
)

Parameters

dwFlags
  [in] Reserved for flag data. Must be zero.

Return Values

Returns S_OK if successful or MS_E_INCOMPATIBLE if the existing sample isn't compatible with the current media stream.

Remarks

Applications do not call this internal method.

IMediaStream::SetSameFormat

IMediaStream Interface

Sets the media stream the same format as a previous stream.

HRESULT SetSameFormat(
  IMediaStream **pStreamThatHasDesiredFormat*,
  DWORD *dwFlags*
  );

Parameters

pStreamThatHasDesiredFormat
  [in] Pointer to a media stream object that has the same format.
dwFlags
  [in] Reserved for flag data. Must be zero.

Return Values

Returns S_OK if successful or E_POINTER if one of the parameters is invalid.

IStreamSample Interface

The **IStreamSample** interface provides control over the behavior of stream samples. You can retrieve the media stream that created the sample, set or retrieve sample start and stop times, check the sample's completion status, and perform a developer-specified function on the sample itself.

When to Implement

Implement this interface when you implement a media stream for a new media type. The interface is exposed on sample objects created by media streams.

When to Use

Use this interface when you want to control data samples created by IMediaStream or its derived interfaces.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IStreamSample methods	**Description**
GetMediaStream	Retrieves a pointer to the media stream object that created the current sample.
GetSampleTimes	Retrieves the current sample's start and end times.
SetSampleTimes	Sets the current sample's start and end times.
Update	Performs a synchronous or an asynchronous update on the current sample.
CompletionStatus	Retrieves the status of the current sample's latest asynchronous update. If the update isn't complete, you can force it to complete.

IStreamSample::CompletionStatus

IStreamSample Interface

Retrieves the status of the current sample's latest asynchronous update. If the update isn't complete, you can force it to complete.

HRESULT CompletionStatus(
  DWORD *dwFlags***,**
  DWORD *dwMilliseconds*
)

Parameters

dwFlags
[in] Value that specifies whether to forcibly complete the update. This value is a combination of one or more of the following flags.

Value	Meaning
COMPSTAT_NOUPDATEOK (0x01)	Force the update to complete as soon as possible, even if the sample update isn't yet complete. If the sample is updating and you didn't set the COMPSTAT_WAIT flag, the method returns MS_S_PENDING. If the sample is waiting to be updated, this method removes it from the queue and returns MS_S_NOTUPDATED.
COMPSTAT_WAIT (0x02)	Wait until the sample finishes updating before returning from this method.
COMPSTAT_ABORT (0x04)	Forces the update to complete, even if it's currently updating. This leaves the sample data in an undefined state. Combine this value with the COMPSTAT_WAITFORCOMPLETION flag to ensure that the update canceled.

dwMilliseconds
[in] If the *dwFlags* parameter is COMPSTAT_WAIT, this value is the number of milliseconds to wait for the update to complete. Specify INFINITE to indicate that you want to wait until the sample updates before this call returns.

Return Values

Returns one of the following values.

Value	Meaning
E_ABORT	The update aborted.
MS_S_ENDOFSTREAM	The sample wasn't updated because it reached the end of the stream.
MS_S_NOUPDATE	The update was forcibly completed; the sample was not updated by the stream.
MS_S_PENDING	An asynchronous update is pending.
S_OK	Success.

IStreamSample::GetMediaStream

IStreamSample Interface

Retrieves a pointer to the media stream object that created the current sample.

HRESULT GetMediaStream(
  IMediaStream** *ppMediaStream*
)

Parameters

ppMediaStream
: [in] Address of a pointer to an IMediaStream interface that will point to the media stream that created the current sample.

Return Values

Returns S_OK if successful or E_POINTER if *ppMediaStream* is invalid.

Remarks

If successful, this method increments the reference count of the media stream specified by *ppMediaStream*.

IStreamSample::GetSampleTimes

IStreamSample Interface

Retrieves the current sample's start and end times. If the sample is updating, this method returns the times after the update completes.

HRESULT GetSampleTimes(
  STREAM_TIME* *pStartTime*,

STREAM_TIME* *pEndTime*,
STREAM_TIME* *pCurrentTime*
)

Parameters

pStartTime
[out] Pointer to a STREAM_TIME value that will contain the sample's start time.
pEndTime
[out] Pointer to a STREAM_TIME value that will contain the sample's end time.
pCurrentTime
[out] Pointer to a STREAM_TIME value that will contain the media stream's current media time.

Return Values

Returns S_OK if successful or E_POINTER if one of the parameters is invalid.

Remarks

For streams that have a clock, the start and end times will be relative to the stream's current time. If the stream doesn't have a clock, the times are media-relative and the current time will be zero.

The *pCurrentTime* parameter enables you to conveniently track the media stream's current time, so you don't have to call IMultiMediaStream::GetTime. Unlike GetTime, however, this method returns S_OK if the stream doesn't have a clock; **GetTime** returns S_FALSE. The value assigned to *pCurrentTime* is the same as the value produced by the following code fragment.

```
pSample->GetMediaStream(&pMediaStream);
pMediaStream->GetMultiMediaStream(&pMultiMediaStream);
pMediaStream->Release();
pMultiMediaStream->GetTime(&pCurrentTime);
pMultiMediaStream->Release();
```

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IStreamSample::SetSampleTimes

IStreamSample Interface

Sets the current sample's start and end times. You can call this method prior to updating the sample.

HRESULT SetSampleTimes(
const STREAM_TIME* *pStartTime*,
const STREAM_TIME* *pEndTime*
)

Parameters

pStartTime
[in] Pointer to a STREAM_TIME value that contains the sample's new start time.
pEndTime
[in] Pointer to a STREAM_TIME value that contains the sample's new end time.

Return Values

Returns S_OK if successful or E_POINTER if one of the parameters is NULL.

Remarks

For streams that have a clock, the times must be relative to the stream's current time. If the stream doesn't have a clock, the times should be relative to the media.

This method only applies to writable streams.

IStreamSample::Update

IStreamSample Interface

Performs a synchronous or an asynchronous update on the current sample.

HRESULT Update(
DWORD *dwFlags*,
HANDLE *hEvent*,
PAPCFUNC *pfnAPC*,
DWORD *dwAPCData*
)

Parameters

dwFlags
[in] Flag that specifies whether the update is synchronous or asynchronous. The SSUPDATE_ASYNC flag specifies an asynchronous update, which you can set if both *hEvent* and *pfnAPC* are NULL. Use SSUPDATE_CONTINUOUS to continuously update the sample until you call the IStreamSample::CompletionStatus method.

hEvent
[in] Handle to an event that this method will trigger when the update is complete.

pfnAPC
[in] Pointer to a Win32 asynchronous procedure call (APC) function that this method will call after it completes the sample update.

dwAPCData
[in] Value that this method passes to the function specified by the *pfnAPC* parameter.

Return Values

Returns one of the following values.

Value	**Meaning**
E_ABORT	The update aborted.
E_INVALIDARG	One of the parameters is invalid.
E_POINTER	One of the parameters is invalid.
MS_E_BUSY	This sample already has a pending update.
MS_S_ENDOFSTREAM	Reached the end of the stream; the sample wasn't updated.
MS_S_PENDING	The asynchronous update is pending.
S_OK	Success.

Remarks

This method can be used to perform a synchronous or asynchronous update of a sample. If both *hEvent* and *pfnAPC* are NULL then the update will be synchronous unless either of the SSUPDATE_ASYNC or SSUPDATE_CONTINUOUS flags is specified. When a synchronous update returns, the result of the function contains the I/O completion status.

You can't specify values for both *hEvent* and *pfnAPC*; the method will fail.

Asynchronous updates might complete before the update returns; in that case, the return value is S_OK. If you specify an event and the update returns S_OK, this method sets the event on return. If you specify an APC function and the update returns S_OK, the APC will not be queued and the function will not be called.

Asynchronous updates that don't complete prior to returning will return a value of MS_S_PENDING.

Applications that create multiple streams must read from each of them to avoid having their data blocked.

IDirectDrawMediaStream Interface

The **IDirectDrawMediaStream** interface controls media streams that appear on Microsoft DirectDraw® surfaces. To stream to a DirectDraw surface, DirectDraw must support the video stream format.

For sample code that implements the multimedia streaming interfaces see Multimedia Streaming Sample Code.

When to Implement

This interface isn't intended for implementation by application developers. It is exposed on DirectDraw media streams that can be added to a DirectShow multimedia stream.

When to Use

Use this interface when you want to output a video stream to a DirectDraw surface.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDirectDrawMediaStream methods	Description
GetFormat	Retrieves the current media stream's format and, optionally, its desired format.
SetFormat	Sets the current media stream's format. If the stream already has allocated samples and the sample format doesn't match the specified format, this method fails.
GetDirectDraw	Retrieves a pointer to the DirectDraw object used by the current media stream.
SetDirectDraw	Sets the current media stream's DirectDraw object.
CreateSample	Creates a stream sample using the specified DirectDraw surface object.
GetTimePerFrame	Retrieves the average frames per second from a video stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawMediaStream::CreateSample

IDirectDrawMediaStream Interface

Creates a stream sample using the specified DirectDraw surface object.

HRESULT CreateSample(
 IDirectDrawSurface* *pSurface*,
 const RECT **pRect*,
 DWORD *dwFlags*,
 IDirectDrawStreamSample** *ppSample*
)

Parameters

pSurface
 [in] Pointer to an existing DirectDraw surface.
pRect
 [in] Pointer to the clipping rectangle you want to use with the specified surface. Set this parameter to NULL if you want to use the entire surface.
dwFlags
 [in] Flag that specifies whether the render should be progressive. To perform a progressive render, set this value to DDSFF_PROGRESSIVERENDER. If you don't set this flag, sample updates will copy the sample data to the surface.
ppSample
 [out] Address of a pointer to an IDirectDrawStreamSample interface that will point to the newly created sample.

Return Values

Returns one of the following values.

Value	**Meaning**
DDERR_INVALIDPIXELFORMAT	The specified pixel format is incompatible with the stream format.
DDERR_INVALIDRECT	The specified clipping rectangle is invalid.
DDERR_INVALIDSURFACETYPE	The specified surface is incompatible with the stream format.
E_POINTER	One or more of the required parameters is invalid.
MS_E_SAMPLEALLOC	The stream already has allocated samples and the surface doesn't match the sample format.
S_OK	Success.

Remarks

This method creates a sample from the current stream and attaches the sample to this surface.

If the stream doesn't have an allocated surface and the specified surface doesn't match the stream's format, this method calls the IDirectDrawMediaStream::SetFormat method on the stream so the two will match.

To perform a progressive render, create a single sample and repeatedly use that sample for successive frames of video. Video decompressors use this technique to do partial updates to the previous frame.

Note, the *pRect* parameter should match the format of the stream (see

IDirectDrawMediaStream::GetFormat). If the wrong clip rectangle is set or no clip rectangle is set, and the surface size does not match the movie size, the movie may not play. If a primary surface is used it is advisable to use a clipping rectangle because the primary surface size may change if the user changes their display settings.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawMediaStream::GetDirectDraw

IDirectDrawMediaStream Interface

Retrieves a pointer to the DirectDraw object used by the current media stream.

HRESULT GetDirectDraw(
IDirectDraw** *ppDirectDraw*
)

Parameters

ppDirectDraw
[out] Address of a pointer to an IDirectDraw interface that will contain the current media stream's associated DirectDraw object.

Return Values

Returns S_OK if successful or E_POINTER if the parameter is invalid.

Remarks

If you haven't initialized the stream yet, the retrieved pointer will be NULL and the method will return S_OK. If you have initialized the stream, this method increments the retrieved pointer's reference count.

Previous Home Topic Contents Index Next

IDirectDrawMediaStream::GetFormat

IDirectDrawMediaStream Interface

Retrieves the current media stream's format and, optionally, its desired format.

HRESULT GetFormat(
 DDSURFACEDESC **pDDSDCurrent*,
 IDirectDrawPalette ***ppDirectDrawPalette*,
 DDSURFACEDESC **pDDSDDesired*,
 DWORD* *pdwFlags*
)

Parameters

pDDSDCurrent
 [out] Pointer to a DirectDraw surface description that will contain the current media stream's format.
ppDirectDrawPalette
 [out] Address of a pointer to an **IDirectDrawPalette** interface if one exists.
pDDSDDesired
 [out] Pointer to a DirectDraw surface description that will contain the current media stream's desired format.
pdwFlags
 [out] Pointer to the flags set in a DDSURFACEDESC structure. Fields of interest include.

Value	**Meaning**
DDSD_HEIGHT	Indicates that the height member of the structure is valid.
DDSD_WIDTH	Indicates that the width member of the structure is valid.
DDSD_PIXELFORMAT	Indicates that the pixel format member of the structure is valid.
DDSD_CAPS	Indicates that the surface capability member of the structure is valid.

Return Values

Returns one of the following values.

Value	**Meaning**
DDERR_INVALIDPARAMS	One of the DirectDraw surface parameters is invalid.
E_POINTER	One or more of the required parameters is invalid.
S_OK	Success.

Remarks

After you call this method, you can either conform to the current format or attempt to change the format by calling the IDirectDrawMediaStream::SetFormat method.

All of this method's parameters are optional; set any of them to NULL to indicate that you don't want to retrieve that information.

To perform a progressive render, create a single sample and repeatedly use that sample for successive frames of video. Video decompressors use this technique to do partial updates to the previous frame.

You must initialize the **dwSize** member of the DDSURFACEDESC structure before calling this method.

The DDSD_CAPS flag will return one of the values listed in the DDSCAPS structure or DDSCAPS_DATAEXCHANGE, which is defined as DDSCAPS_SYSTEMMEMORY|DDSCAPS_VIDEOMEMORY in Ddrawex.h.

Previous Home Topic Contents Index Next

IDirectDrawMediaStream::GetTimePerFrame

IDirectDrawMediaStream Interface

Retrieves the average frames per second from a video stream.

HRESULT GetTimePerFrame(
 STREAM_TIME **pFrameTime*
)

Parameters

pFrameTime
: [out] STREAM_TIME value that indicates the average time per frame in 100-nanosecond units.

Return Values

Returns S_OK if successful or E_POINTER if the pointer is invalid.

IDirectDrawMediaStream::SetDirectDraw

IDirectDrawMediaStream Interface

Sets the current media stream's DirectDraw object.

HRESULT SetDirectDraw(

IDirectDraw* *pDirectDraw*
)

Parameters

pDirectDraw
: [in] Address of a pointer to an IDirectDraw interface that contains the media stream's new DirectDraw object.

Return Values

Returns S_OK if successful.

Remarks

This method fails if the current stream already has allocated samples and its DirectDraw object differs from the specified one. It will always succeed if the specified DirectDraw object matches the stream's current object.

If this stream has no allocated samples, you can set *pDirectDraw* to NULL. This forces the stream to release its reference to the current DirectDraw object.

IDirectDrawMediaStream::SetFormat

IDirectDrawMediaStream Interface

Sets the format of the current media stream.

HRESULT SetFormat(
const DDSURFACEDESC **pDDSurfaceDesc*,
IDirectDrawPalette **pDirectDrawPalette*
)

Parameters

pDDSurfaceDesc
: [in] Pointer to a DirectDraw surface description that contains the new format.

pDirectDrawPalette
: [in] Optional parameter that is a pointer to an IDirectDrawPalette interface.

Return Values

Returns one of the following values.

Value	Meaning
S_OK	Success.
DDERR_INVALIDSURFACETYPE	The specified format is incompatible with the current stream.
MS_E_SAMPLEALLOC	Can't change the format because one or more stream samples are already allocated for this stream.

Remarks

If the stream already has allocated samples and the sample format doesn't match the specified format, this method fails. This method always succeeds if the specified format matches the current format.

IDirectDrawStreamSample Interface

The **IDirectDrawStreamSample** interface provides methods that set and retrieve pointers to the DirectDraw surface associated with the current stream sample.

When to Implement

This interface isn't intended for implementation by application developers. It is exposed by sample objects created by the DirectDraw stream.

When to Use

Use this interface when applications need to set clipping rectangles and retrieve the rendering surface for DirectDraw stream samples.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDirectDrawStreamSample methods	Description
GetSurface	Retrieves pointers to the current sample's DirectDraw surface and associated clipping rectangle.
SetRect	Changes the clipping rectangle for a sample.

Previous Home Topic Contents Index Next

IDirectDrawStreamSample::GetSurface

IDirectDrawStreamSample Interface

Retrieves pointers to the current sample's DirectDraw surface and associated clipping rectangle.

HRESULT GetSurface(
  IDirectDrawSurface ** *ppDirectDrawSurface*,
  RECT * *pRect*
  );

Parameters

ppDirectDrawSurface
: [out] Address of a pointer to an IDirectDrawSurface interface that specifies the sample's new surface. Set this parameter to NULL if you don't want to specify a new surface.

pRect
: [out] Pointer to a RECT structure that will contain the current sample's clipping rectangle. Set this parameter to NULL if you don't want to retrieve the clipping rectangle.

Return Values

Returns S_OK if successful.

Remarks

Both parameters are optional. All implementations of this interface must support null values as valid parameters. If you retrieve a surface pointer, this method increments its reference count, so you must release the reference.

IDirectDrawStreamSample::SetRect

IDirectDrawStreamSample Interface

Changes the clipping rectangle for a sample.

HRESULT SetRect (
 const RECT * *pRect*
)

Parameters

pRect
: [in] Pointer to a RECT structure that specifies the stream's new clipping rectangle.

Return Values

Returns one of the following values.

Value	Meaning
DDERR_INVALIDPIXELFORMAT	The stream isn't compatible with the pixel format.
DDERR_INVALIDRECT	The specified rectangle is invalid.
DDERR_INVALIDSURFACETYPE	The stream isn't compatible with the surface.
E_POINTER	One of the pointers is invalid.
MS_E_SAMPLEALLOC	The stream format doesn't match the surface and samples are currently allocated to the stream.
S_OK	Success.

Remarks

Both parameters are optional; set either to NULL to avoid changing that value. If the surface format doesn't match the stream format, this method fails.

If the new rectangle's size isn't the same as the current rectangle, a call to this method will fail.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAudioMediaStream Interface

The **IAudioMediaStream** interface controls audio media streams by providing methods that set and get the stream's format. This interface inherits from the IMediaStream interface and is used to create one or more IAudioStreamSample objects. You can also use it to set and retrieve the stream data's current format.

This interface is currently defined only for PCM format audio data.

For sample code that implements the audio streaming interfaces, see Multimedia Streaming Sample Code.

When to Implement

Like video, audio is contained in a self-describing container object. Implement this interface when an object needs to control streaming audio.

When to Use

Use this interface when you want to generate audio in your application.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IAudioMediaStream methods	Description
GetFormat	Retrieves the stream data's current format.
SetFormat	Sets the format for the stream.
CreateSample	Creates an audio stream sample for use with this stream.

IAudioMediaStream::CreateSample

IAudioMediaStream Interface

Creates an audio stream sample for use with the specified stream.

HRESULT CreateSample(
  IAudioData **pAudioData*,
  DWORD *dwFlags*,
  IAudioStreamSample ***ppSample*
  );

Parameters

pAudioData
[in] Pointer to an IAudioData container. **IAudioData** objects can be referenced by samples in more than one stream.

dwFlags
 [in] Reserved for flag data. Must be zero.
ppSample
 [out] Address of a pointer to the new IAudioStreamSample interface.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Remarks

The *pAudioData* object defines the data's format.

IAudioMediaStream::GetFormat

IAudioMediaStream Interface

Retrieves the stream data's current format.

HRESULT GetFormat(
 WAVEFORMATEX **pWaveFormatCurrent* **);**

Parameters

pWaveFormatCurrent
 [out] Pointer to a WAVEFORMATEX structure that contains the stream data's current format.

Return Values

Returns S_OK if successful or E_POINTER if the required parameter is NULL.

Remarks

Currently, DirectShow only supports PCM wave data.

IAudioMediaStream::SetFormat

IAudioMediaStream Interface

Sets the format for the stream.

HRESULT SetFormat(
  const WAVEFORMATEX **lpWaveFormat*
  );

Parameters

lpWaveFormat
[in] Pointer to a WAVEFORMATEX structure that contains stream format information.

Return Values

Returns an HRESULT value, which can include the following values or others not listed.

Value	Meaning
MS_E_INCOMPATIBLE	Format of the IAudioData object is not compatible with stream.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAudioStreamSample Interface

The **IAudioStreamSample** interface retrieves information from the underlying IAudioData data objects.

For sample code that implements the audio streaming interfaces, see Multimedia Streaming Sample Code.

When to Implement

Implement this interface on audio stream sample objects when they need access to an IAudioData object's data .

When to Use

Use this interface when your application needs to access an IAudioData object's data for its audio stream.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IAudioStreamSample methods	**Description**
GetAudioData	Retrieves the address of a pointer to the IAudioData object associated with the sample.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAudioStreamSample::GetAudioData

IAudioStreamSample Interface

Retrieves the address of a pointer to the IAudioData object associated with the sample.

HRESULT GetAudioData(
  IAudioData ***ppAudio* **);**

Parameters

ppAudio
    [out] Address of a pointer to the IAudioData object.

Return Values

Returns S_OK if successful or E_POINTER if the parameter is NULL.

Previous Home Topic Contents Index Next

IMemoryData Interface

The **IMemoryData** interface contains methods that set and retrieve memory data on audio data objects. Audio data objects provide the underlying data which stream samples access. This interface provides a way to initialize memory buffers and to set actual amounts of audio data in the objects. Additionally, the GetInfo method can be used to retrieve audio memory data.

When to Implement

Implement this interface on underlying audio data objects that audio stream sample objects will access.

When to Use

Typically these methods are called by the IAudioMediaStream or IAudioStreamSample object, rather than by the application.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IMemoryData methods	Description
SetBuffer	Initializes a memory buffer with a pointer to memory and length.
GetInfo	Retrieves information describing an audio data object.
SetActual	Sets the amount of audio data currently in the object, in bytes.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMemoryData::GetInfo

IMemoryData Interface

Retrieves information describing an audio data object.

HRESULT GetInfo(
  DWORD **pdwLength*,
  BYTE ***ppbData*,
  DWORD **pcbActualData* **);**

Parameters

pdwLength
: [out] Length of memory in bytes, if non-NULL.

ppbData
: [out] Pointer to the memory, if non-NULL.

pcbActualData
: [out] Length of data in bytes, if non-NULL.

Return Values

Returns S_OK if successful.

Remarks

This method determines how much data is actually in the object at the moment as last set by SetActual.

IMemoryData::SetActual

IMemoryData Interface

Sets the amount of audio data currently in the object.

HRESULT SetActual(
  DWORD *cbDataValid*
  );

Parameters

cbDataValid
: [in] Amount of data, in bytes.

Return Values

Returns S_OK if successful or E_POINTER if the required parameter is NULL.

Remarks

This method is usually called by the IAudioMediaStream or IAudioStreamSample object, rather than by the application.

Previous Home Topic Contents Index Next

IMemoryData::SetBuffer

IMemoryData Interface

Initializes a memory buffer with a pointer to memory and length.

HRESULT SetBuffer(
  DWORD *cbSize***,**
  BYTE **pbData***,**
  DWORD *dwFlags* **);**

Parameters

cbSize
    [in] Size of memory pointed to by *pbData*, in bytes.
pbData
    [in] Pointer to memory that this object will use.
dwFlags
    [in] Reserved for flag data. Must be zero.

Return Values

Returns S_OK if successful or E_INVALIDARG if *cbSize* is zero or *pbData* is NULL.

Remarks

This method can be called more than once.

Do not call this method when the IStreamSample::Update method is processing a sample.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAudioData Interface

The **IAudioData** interface provides methods that enable applications to set and get the underlying audio data that audio streams will reference. The audio data format is set in the

WAVEFORMATEX structure.

When to Implement

Implement this interface on underlying audio data objects that audio stream sample objects will access.

When to Use

Applications use this interface to set and retrieve information on underlying data objects that an audio stream will reference.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.
IAudioData methods	**Description**
GetFormat	Retrieves the current data format.
SetFormat	Sets the current data format.

Previous Home Topic Contents Index Next

IAudioData::GetFormat

IAudioData Interface

Retrieves the current data format.

HRESULT GetFormat(
 WAVEFORMATEX **pWaveFormatCurrent* **);**

Parameters

pWaveFormatCurrent
 [out] Pointer to a WAVEFORMATEX structure that contains the current data format.

Return Values

Returns S_OK if successful or E_POINTER if pointer is invalid.

Remarks

Currently, DirectShow only supports PCM wave data.

See Also

IAudioData::SetFormat

Previous Home Topic Contents Index Next

IAudioData::SetFormat

IAudioData Interface

Sets the current data format.

HRESULT SetFormat(
  const WAVEFORMATEX **lpWaveFormat* **);**

Parameters

lpWaveFormat
    [in] Pointer to a WAVEFORMATEX structure that will contain the current data format.

Return Values

Returns an HRESULT value, which can include the following values.

Value	**Meaning**
E_POINTER	Invalid pointer argument.
E_INVALIDARG	Invalid format.
S_OK	Success.

See Also

IAudioData::GetFormat

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMMultiMediaStream Interface

The **IAMMultiMediaStream** interface exposes Microsoft® DirectShow™ functionality to multimedia stream developers. You can use its methods to automatically generate filter graphs, open files or monikers for playback or capture of incoming data, and render a given filter graph.

When to Implement

Implement this interface when you want to provide multimedia stream-based support for DirectShow media types in your applications.

When to Use

Use this interface when you want to control DirectShow-supported media playback in your multimedia streaming applications.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IAMMultiMediaStream methods	**Description**
Initialize	Sets the stream type. If the *pFilterGraph* parameter is non-NULL the filter graph passed in is used for the stream.
GetFilterGraph	Retrieves the associated filter graph's IGraphBuilder interface.
GetFilter	Retrieves the specified filter from the current filter graph.
AddMediaStream	Adds the specified media stream to the current filter graph.
OpenFile	Opens and automatically creates a filter graph for the specified media file. If DirectShow doesn't support the file format, this method does nothing.
OpenMoniker	Opens a file or device moniker; you can read media data from this moniker if DirectShow supports the media type.
Render	Renders the current filter graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMMultiMediaStream::AddMediaStream

IAMMultiMediaStream Interface

Adds the specified media stream to the current filter graph.

HRESULT AddMediaStream(
IUnknown* *pStreamObject*,
const MSPID **pPurposeID*,
DWORD *dwFlags*,
IMediaStream** *ppNewStream*
)

Parameters

pStreamObject
[in] Pointer to an IUnknown interface that points to either the media stream or underlying stream object you want to add to the current filter graph.
pPurposeID
[in] Pointer to the purpose ID for the newly added media stream.
dwFlags
[in] Value that modifies the media stream's behavior; it is a combination of one or more of the following values.

AMMSF_ADDDEFAULTRENDERER	Add a default renderer.
AMMSF_CREATEPEER	Create a peer stream based on the same object as a pStreamObject.

ppNewStream
[out] Address of a pointer to an IMediaStream interface that will point to the newly added media stream. This parameter is optional.

Return Values

Returns an HRESULT value, which can include the following values:

Value	**Meaning**
MS_E_PURPOSEID	Stream being added has a different purpose ID from the one specified or a stream with the specified purpose ID already exists.
E_POINTER	Null pointer argument.
S_OK	Success.

Remarks

If dwFlags specifies AMMSF_ADDDEFAULTRENDERER then the default renderer for the given purpose Id is created if possible. Currently the only default renderer supported is for audio using DirectSound. In this case the *pStreamObject* parameter must be NULL and any calls to the IMultiMediaStream::GetMediaStream or IMultiMediaStream::EnumMediaStreams methods will not recognize the stream.

If *dwFlags* specifies AMMSF_CREATEPEER then a Media Stream is created using pStreamObject and added to the current multimedia stream. pStreamObject varies depending on the stream type. In general pStreamObject can point to an IMediaStream object, in which case a stream with the sample purpose ID and format is created. For IDirectDraw streams it can also be a pointer to an **IDirectDraw** object.

If neither flag is set then pStreamObject can be one of the following:

An IAMMediaStream object	This stream is then added to the streams in the multimedia stream.
NULL	In this case a default IMediaStream object is added to the stream with a default underlying object if required.
A pointer to an underlying object	This is used to construct default streams. For video streams this can be an IDirectDraw pointer.

IAMMultiMediaStream::GetFilter

IAMMultiMediaStream Interface

Retrieves the specified filter from the current filter graph.

HRESULT GetFilter(
 IMediaStreamFilter** *ppFilter*
)

Parameters

ppFilter
 [out] Address of a pointer to an **IMediaStreamFilter** interface that will point to the current filter.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

IAMMultiMediaStream::GetFilterGraph

IAMMultiMediaStream Interface

Retrieves the associated filter graph's IGraphBuilder interface.

HRESULT GetFilterGraph(
  IGraphBuilder ***ppGraphBuilder*
  );

Parameters

ppGraphBuilder
    [out] Address of a pointer to an *IGraphBuilder* interface that will point to the current filter graph.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Previous Home Topic Contents Index Next

IAMMultiMediaStream::Initialize

IAMMultiMediaStream Interface

Sets the stream type. If the pFilterGraph parameter is non-NULL the filter graph passed in is used for the stream.

HRESULT Initialize(
  STREAM_TYPE *StreamType*,
  DWORD *dwFlags*,
  IGraphBuilder * *pFilterGraph*
)

Parameters

StreamType
    [in] STREAM_TYPE enumeration value that specifies the new filter graph's stream type.
dwFlags
    [in] Either contains the AMMSF_NOGRAPHTHREAD flag, which creates a filter graph object on the current thread, or zero.
pFilterGraph
    [in] [optional] Address of an IGraphBuilder interface that will point to the new filter graph. This parameter is optional; only pass in a valid pointer if you must access the filter graph at a later time.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Remarks

Using the AMMSF_NOGRAPHTHREAD flag is safe provided the current thread does not exit before the stream object is released by the application and the current thread processes window messages.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMMultiMediaStream::OpenFile

IAMMultiMediaStream Interface

Opens and automatically creates a filter graph for the specified media file. If DirectShow doesn't support the file format, this method does nothing.

HRESULT OpenFile(
  LPCWSTR *pszFileName*,
  DWORD *dwFlags*
)

Parameters

pszFileName
    [in] Name of the file you want to open.
dwFlags
    [in] Value that modifies how the filter graph will render the specified file. This value is a combination of one or more of the following flags:

Value	**Meaning**
AMMSF_RENDERTOEXISTING	Only render to existing streams
AMMSF_RENDERALLSTREAMS	Render all streams, including those that do not have an existing media stream.
AMMSF_NORENDER	Open the file, but do not render any streams. This flag should always be accompanied with the AMMSF_RUN flag.
AMMSF_NOCLOCK	Run the stream with no clock.
AMMSF_RUN	Set the stream into the run state.

Return Values

Returns one of the following values.

Value	Meaning
E_INVALIDARG	The *dwFlags* parameter is invalid.
E_POINTER	This method tried to access an invalid pointer.
S_OK	Success.

Remarks

The AMMSF_RENDERALLSTREAMS flag will create default rendering filters for video and audio if they do not exist. However, these default filters cannot be accessed by the IStreamSample::GetMediaStream method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMMultiMediaStream::OpenMoniker

IAMMultiMediaStream Interface

Opens a file or device moniker; you can read media data from this moniker if DirectShow supports the moniker.

HRESULT OpenMoniker(
  IBindCtx **pCtx*,
  IMoniker* *pMoniker*,
  DWORD *dwFlags*
)

Parameters

pCtx
: [in] Pointer to the bind context associated with the moniker.

pMoniker
: [in] Pointer to an IMoniker interface that specifies the moniker you want to open.

dwFlags
: [in] Value that modifies how the filter graph will render the specified file. This value is a combination of one or more of the following flags:

Value	Meaning
AMMSF_RENDERTOEXISTING	Only render to existing streams
AMMSF_RENDERALLSTREAMS	Render all streams, including those that do not have an existing media stream.
AMMSF_NORENDER	Open the file, but do not render any streams. This flag should always be accompanied with the AMMSF_RUN flag.
AMMSF_NOCLOCK	Run the stream with no clock.

AMMSF_RUN	Set the stream into the run state.

Return Values

Returns S_OK if successful or E_INVALIDARG if the *dwFlags* parameter is invalid.

IAMMultiMediaStream::Render

IAMMultiMediaStream Interface

Renders the current filter graph.

HRESULT Render(
  DWORD *dwFlags*
)

Parameters

dwFlags
[in] Value that specifies how the filter graph renders the current multimedia stream. This value currently must be AMMSF_NOCLOCK.

Return Values

Returns S_OK if successful or E_INVALIDARG if the *dwFlags* parameter is invalid.

Remarks

This method renders each of the source streams for a stream of type STREAMTYPE_WRITE. This can be called several times, for instance, each time a source stream is added, the stream is not set into running mode. Use the IMultiMediaStream::SetState method to set the stream into running mode after calling this method.

The AMMSF_RENDERALLSTREAMS flag will create default rendering streams for video and audio if they do not exist.

IAMMediaStream Interface

The **IAMMediaStream** interface handles the internal connections between DirectShow filters and filter graphs in applications that use multimedia streaming. This enables applications to automatically negotiate the transfer and conversion of data from the source to the application without having to write code to handle the connection, transfer of data, data conversion, and actual data rendering or file storage. This provides a uniform and predictable method of data access and control.

When to Implement

This interface isn't intended for implementation by application developers.

When to Use

This interface isn't intended for use by application developers.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IAMMediaStream methods	**Description**
Initialize	Creates and initializes a new media stream with the specified stream type and purpose ID.
SetState	Sets the filter state.
JoinAMMultiMediaStream	The IAMMultiMediaStream::AddMediaStream method calls this method, which adds the specified media stream to the current multimedia stream.
JoinFilter	Connects a media stream to a media stream filter in the underlying filter graph.
JoinFilterGraph	Connects a media stream filter to a filter graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMMediaStream::Initialize

IAMMediaStream Interface

Creates and initializes a new media stream with the specified stream type and purpose ID.

HRESULT Initialize(
  IUnknown **pSourceObject***,**
  DWORD *dwFlags***,**
  REFMSPID *PurposeID***,**
  STREAM_TYPE *StreamType*
)

Parameters

pSourceObject
  [in] Pointer to an IUnknown source object.
dwFlags
  [in] Flags. Must be zero.
PurposeID
  [in] Purpose ID for the new media stream.
StreamType
  [in] A STREAM_TYPE enumeration value that specifies the new media stream's media type.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are invalid.

IAMMediaStream::JoinAMMultiMediaStream

IAMMediaStream Interface

The IAMMultiMediaStream::AddMediaStream method calls this method, which adds the specified media stream to the current multimedia stream.

HRESULT JoinAMMultiMediaStream(
  IAMMultiMediaStream* *pAMMultiMediaStream*
)

Parameters

pAMMultiMediaStream
  [in] Specifies the IAMMultiMediaStream object to add the current media stream to.

Return Values

Returns S_OK if successful or MS_E_SAMPLEALLOC if the media stream already has allocated stream samples.

Remarks

Don't increment the reference count of the supplied multimedia stream because it is already accounted for when created.

Applications should not call this method.

Previous Home Topic Contents Index Next

IAMMediaStream::JoinFilter

IAMMediaStream Interface

Connects a media stream to a media stream filter in the underlying filter graph.

HRESULT JoinFilter( )

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Remarks

Don't increment the reference count for the specified media stream filter.

Applications should not call this method.

Previous Home Topic Contents Index Next

IAMMediaStream::JoinFilterGraph

IAMMediaStream Interface

Connects a media stream filter to a filter graph.

HRESULT JoinFilterGraph(
IFilterGraph* *pGraph*
)

Parameters

pGraph
[in] Indicates the current media stream filter to add to the specified filter graph.

Return Values

Returns S_OK if successful or E_POINTER if one or more of the required parameters are NULL.

Remarks

Don't increment the reference count of the specified filter graph.

Applications should not call this method.

IAMMediaStream::SetState

IAMMediaStream Interface

Sets the filter state.

HRESULT SetState(
FILTER_STATE *State*
)

Parameters

State
[in] Sets the filter's state, as specified by the FILTER_STATE enumerated type.

Return Values

Returns S_OK if successful or E_INVALIDARG if the *State* parameter is invalid..

Remarks

Applications should not call this method.

Multimedia Streaming Data Types

This section describes the Microsoft® DirectShow™ multimedia streaming data types.

Data type	Description
MSPID	Media stream purpose IDs define the purpose of a media stream. A purpose ID is simply typedefed as a GUID.
STREAM_STATE	Describes the state of the stream.
STREAM_TIME	Stream time measured in 100-nanosecond increments. This type is defined to be a 64-bit integer (LONGLONG).
STREAM_TYPE	Defines the direction of data flow for the stream.

STREAM_STATE

Multimedia Streaming Data Types

Describes the state of the stream.

```
typedef enum {
        STREAMSTATE_STOP        = 0,
        STREAMSTATE_RUN         = 1
} STREAM_STATE;
```

Values

STREAMSTATE_STOP
Stop state.
STREAMSTATE_RUN
Run state.

Remarks

Change the state by calling the IMultiMediaStream::SetState method.

Previous Home Topic Contents Index Next

STREAM_TYPE

Multimedia Streaming Data Types

Defines the direction of data flow for the stream.

```
typedef enum {
        STREAMTYPE_READ         = 0,
        STREAMTYPE_WRITE        = 1,
        STREAMTYPE_TRANSFORM= 2
} STREAM_TYPE;
```

Values

STREAMTYPE_READ
: Application can read the stream.

STREAMTYPE_WRITE
: Application can write to the stream.

STREAMTYPE_TRANSFORM
: Application reads and writes the stream.

Remarks

Transform streams are read/write where the sample is updated in place.

Previous Home Topic Contents Index Next

Error and Success Codes for Multimedia Streaming

The following list contains error messages and success notifications for applications that use the multimedia streaming interfaces. This list does not contain all possible errors; the errors shown apply specifically to DirectShow implementation of the multimedia streaming interfaces. Note that success codes start with MS_S and return TRUE from the SUCCEEDED COM macro and FALSE from the COM FAILED macro.

Message	**Hexadecimal code**	**Meaning**
MS_S_PENDING	0x00040001	Sample update is not yet complete.
MS_S_NOUPDATE	0x00040002	Sample was not updated after forced completion.
MS_S_ENDOFSTREAM	0x00040003	End of stream. Sample not updated.
MS_E_SAMPLEALLOC	0x80040401	An IMediaStream object could not be removed from an IMultiMediaStream object because it still contains at least one allocated sample.
MS_E_PURPOSEID	0x80040402	The specified purpose id cannot be used for the call.
MS_E_NOSTREAM	0x80040403	No stream can be found with the specified attributes.
MS_E_NOSEEKING	0x80040404	Seeking not supported for this IMultiMediaStream object.
MS_E_INCOMPATIBLE	0x80040405	The stream formats are not compatible.
MS_E_BUSY	0x80040406	The sample is busy.
MS_E_NOTINIT	0x80040407	The object cannot accept the call because its initialize function or equivalent has not been called.
MS_E_SOURCEALREADYDEFINED	0x80040408	Source already defined.
MS_E_INVALIDSTREAMTYPE	0x80040409	The stream type is not valid for this operation.
MS_E_NOTRUNNING	0x8004040A	The IMultiMediaStream object is not in running state.

Multimedia Streaming Component Objects

This article provides a table that describes the various components that Microsoft® DirectShow™ supports. These objects support the multimedia streaming interfaces. This article also includes a diagram that shows the hierarchy of component objects.

Contents of this article:

- Component Objects Table
- Object References Diagram

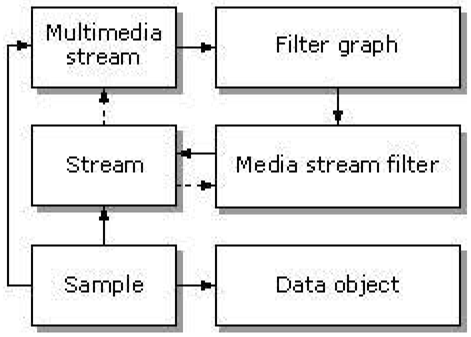
Component Objects Table

This section describes the various components that are supported by DirectShow.

Object	**Description**	**Interfaces supported**
CLSID_AMMultiMediaStream	DirectShow implementation of multimedia stream.	IAMMultiMediaStream, IMultiMediaStream
CLSID_MediaStreamFilter	Provides multimedia streaming functionality for the CLSID_AMMultiMediaStream object through the IAMMultiMediaStream interface.	IBaseFilter
CLSID_AMDirectDrawStream	DirectDraw® media stream that can be added to a DirectShow multimedia stream.	IAMMediaStream, IMediaStream, IDirectDrawMediaStream, IPin, IMemInputPin
Samples created by the DirectDraw stream.		IStreamSample, IDirectDrawStreamSample, IMediaSample
CLSID_ AMMediaTypeStream	Can create media samples for any DirectShow-supported data type	IAMMediaStream, IMediaStream, IPin, IMemInputPin
Samples created by the CLSID_ AMMediaTypeStream object		IStreamSample, IMediaSample, IMediaSample2
CLSID_AMAudioData	Implementation of IAudioData audio container object	IAudioData

Object References Diagram

The hierarchy of objects creates some interesting circular references between the DirectShow objects. The following diagram shows all the objects and their references. Strong references (those that increment the referenced object) are indicated by solid lines. Weak references (those that do not AddRef the referenced object) are indicated by a dotted line.



Samples hold strong references to the multimedia stream object, while the media streams do not.

Multimedia Streaming Sample Code

This article provides sample code that implements the Multimedia Streaming interfaces. The video streaming sample code demonstrates how to read a file and render it to a primary Microsoft® DirectDraw® surface. This code has no error checking; see Use Multimedia Streaming in DirectShow Applications for a more thorough, line by line, description of the video streaming code.

The second code sample demonstrates how to use the audio streaming interfaces to stream audio data.

Contents of this article:

- Video Streaming Sample Code
- Audio Streaming Sample Code

Video Streaming Sample Code

This sample code reads a file and renders it to a primary DirectDraw surface.

```
#include <stdio.h>
#include "ddraw.h"      // DirectDraw interfaces
#include "mmstream.h"   // Multimedia stream interfaces
#include "amstream.h"   // DirectShow multimedia stream interfaces
#include "ddstream.h"   // DirectDraw multimedia stream interfaces


void RenderStreamToSurface(IDirectDrawSurface *pSurface, IMultiMediaStream *pMMStre
{
        IMediaStream *pPrimaryVidStream;
        IDirectDrawMediaStream *pDDStream;
        IDirectDrawStreamSample *pSample;
        RECT rect;
        DDSURFACEDESC ddsd;

        pMMStream->GetMediaStream(MSPID_PrimaryVideo, &pPrimaryVidStream);
        pPrimaryVidStream->QueryInterface(IID_IDirectDrawMediaStream, (void **)&pDI
        ddsd.dwSize = sizeof(ddsd);
        pDDStream->GetFormat(&ddsd, NULL, NULL, NULL);
        rect.top = rect.left = 0;
        rect.bottom = ddsd.dwHeight;
        rect.right = ddsd.dwWidth;
        pDDStream->CreateSample(pSurface, &rect, 0, &pSample);

        pMMStream->SetState(STREAMSTATE_RUN);
        while (pSample->Update(0, NULL, NULL, NULL) == S_OK);
        pMMStream->SetState(STREAMSTATE_STOP);

        pSample->Release();
        pDDStream->Release();
        pPrimaryVidStream->Release();
}

void RenderFileToMMStream(const char * szFileName, IMultiMediaStream **ppMMStream,
{
        IAMMultiMediaStream *pAMStream;
        CoCreateInstance(CLSID_AMMultiMediaStream, NULL, CLSCTX_INPROC_SERVER,
                      IID_IAMMultiMediaStream, (void **)&pAMStream);
        WCHAR   wPath[MAX_PATH];              // Wide (32-bit) string name
        MultiByteToWideChar(CP_ACP, 0, szFileName, -1, wPath,
                                sizeof(wPath)/sizeof(wPath[0]));

        pAMStream->Initialize(STREAMTYPE_READ, AMMSF_NOGRAPHTHREAD, NULL);
        pAMStream->AddMediaStream(pDD, &MSPID_PrimaryVideo, 0, NULL);
        pAMStream->AddMediaStream(NULL, &MSPID_PrimaryAudio, AMMSF_ADDDEFAULTRENDE
        pAMStream->OpenFile(wPath, 0);
        *ppMMStream = pAMStream;
}

int _CRTAPI1 main(int argc, char *argv[])
{
        if (argc < 2) {
        printf("Usage : showstrm movie.ext\n");
        exit(0);}

        DDSURFACEDESC ddsd;
        IDirectDraw *pDD;
        IDirectDrawSurface *pPrimarySurface;
        IMultiMediaStream *pMMStream;

        CoInitialize(NULL);

        DirectDrawCreate(NULL, &pDD, NULL);
        pDD->SetCooperativeLevel(GetDesktopWindow(), DDSCL_NORMAL);
        ddsd.dwSize = sizeof(ddsd);
  ddsd.dwFlags = DDSD_CAPS;
        ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;
        pDD->CreateSurface(&ddsd, &pPrimarySurface, NULL);
        RenderFileToMMStream(argv[1], &pMMStream, pDD);
        RenderStreamToSurface(pPrimarySurface, pMMStream);
        pMMStream->Release();
        pPrimarySurface->Release();
        pDD->Release();

        CoUninitialize();
        return 0;
}
```

Audio Streaming Sample Code

The following code sample demonstrates how to stream audio data using the IAudioMediaStream, IAudioStreamSample, IMemoryData, and IAudioData interfaces.

```
#include <windows.h>
#include <mmsystem.h>
#include <amstream.h>

/******************************************************************

   Trivial wave player stuff

 ******************************************************************/
```

```
class CWaveBuffer;

class CWaveBuffer {
    public:
        CWaveBuffer();
        ~CWaveBuffer();
        BOOL Init(HWAVEOUT hWave, int Size);
        void Done();
        BOOL Write(PBYTE pData, int nBytes, int& BytesWritten);
        void Flush();

    private:
        WAVEHDR      m_Hdr;
        HWAVEOUT     m_hWave;
        int          m_nBytes;
};

class CWaveOut {
    public:
        CWaveOut(LPCWAVEFORMATEX Format, int nBuffers, int BufferSize);
        ~CWaveOut();
        void Write(PBYTE Data, int nBytes);
        void Flush();
        void Wait();
        void Reset();
    private:
        const HANDLE       m_hSem;
        const int          m_nBuffers;
        int          m_CurrentBuffer;
        BOOL         m_NoBuffer;
        CWaveBuffer *m_Hdrs;
        HWAVEOUT     m_hWave;
};

/*
    CWaveBuffer
*/

CWaveBuffer::CWaveBuffer()
{
}

BOOL CWaveBuffer::Init(HWAVEOUT hWave, int Size)
{
    m_hWave  = hWave;
    m_nBytes = 0;

    /*  Allocate a buffer and initialize the header */
    m_Hdr.lpData = (LPSTR)LocalAlloc(LMEM_FIXED, Size);
    if (m_Hdr.lpData == NULL) {
        return FALSE;
    }
    m_Hdr.dwBufferLength  = Size;
    m_Hdr.dwBytesRecorded = 0;
    m_Hdr.dwUser = 0;
    m_Hdr.dwFlags = 0;
    m_Hdr.dwLoops = 0;
    m_Hdr.lpNext = 0;
    m_Hdr.reserved = 0;

    /*  Prepare it */
    waveOutPrepareHeader(hWave, &m_Hdr, sizeof(WAVEHDR));
    return TRUE;
}

CWaveBuffer::~CWaveBuffer() {
    if (m_Hdr.lpData) {
        waveOutUnprepareHeader(m_hWave, &m_Hdr, sizeof(WAVEHDR));
        LocalFree(m_Hdr.lpData);
    }
}

void CWaveBuffer::Flush()
{
    //ASSERT(m_nBytes != 0);
    m_nBytes = 0;
    waveOutWrite(m_hWave, &m_Hdr, sizeof(WAVEHDR));
}

BOOL CWaveBuffer::Write(PBYTE pData, int nBytes, int& BytesWritten)
{
    //ASSERT((DWORD)m_nBytes != m_Hdr.dwBufferLength);
    BytesWritten = min((int)m_Hdr.dwBufferLength - m_nBytes, nBytes);
    CopyMemory((PVOID)(m_Hdr.lpData + m_nBytes), (PVOID)pData, BytesWritten);
    m_nBytes += BytesWritten;
    if (m_nBytes == (int)m_Hdr.dwBufferLength) {
        /*  Write it! */
        m_nBytes = 0;
        waveOutWrite(m_hWave, &m_Hdr, sizeof(WAVEHDR));
        return TRUE;
    }
    return FALSE;
}

void CALLBACK WaveCallback(HWAVEOUT hWave, UINT uMsg, DWORD dwUser, DWORD dw1, DWOR
{
    if (uMsg == WOM_DONE) {
        ReleaseSemaphore((HANDLE)dwUser, 1, NULL);
    }
}

/*
    CWaveOut
*/

CWaveOut::CWaveOut(LPCWAVEFORMATEX Format, int nBuffers, int BufferSize) :
    m_nBuffers(nBuffers),
    m_CurrentBuffer(0),
    m_NoBuffer(TRUE),
    m_hSem(CreateSemaphore(NULL, nBuffers, nBuffers, NULL)),
    m_Hdrs(new CWaveBuffer[nBuffers]),
    m_hWave(NULL)
{
    /*  Create wave device */
    waveOutOpen(&m_hWave,
                WAVE_MAPPER,
                Format,
                (DWORD)WaveCallback,
                (DWORD)m_hSem,
                CALLBACK_FUNCTION);

    /*  Initialize the wave buffers */
    for (int i = 0; i < nBuffers; i++) {
        m_Hdrs[i].Init(m_hWave, BufferSize);
    }
}
```

```
CWaveOut::~CWaveOut()
{
    /*  First get our buffers back */
    waveOutReset(m_hWave);

    /*  Free the buffers */
    delete [] m_Hdrs;

    /*  Close the wave device */
    waveOutClose(m_hWave);

    /*  Free our semaphore */
    CloseHandle(m_hSem);
}

void CWaveOut::Flush()
{
    if (!m_NoBuffer) {
        m_Hdrs[m_CurrentBuffer].Flush();
        m_NoBuffer = TRUE;
        m_CurrentBuffer = (m_CurrentBuffer + 1) % m_nBuffers;
    }
}

void CWaveOut::Reset()
{
    waveOutReset(m_hWave);
}


void CWaveOut::Write(PBYTE pData, int nBytes)
{
    while (nBytes != 0) {
        /*  Get a buffer if necessary */
        if (m_NoBuffer) {
            WaitForSingleObject(m_hSem, INFINITE);
            m_NoBuffer = FALSE;
        }

        /*  Write into a buffer */
        int nWritten;
        if (m_Hdrs[m_CurrentBuffer].Write(pData, nBytes, nWritten)) {
            m_NoBuffer = TRUE;
            m_CurrentBuffer = (m_CurrentBuffer + 1) % m_nBuffers;
            nBytes -= nWritten;
            pData += nWritten;
        } else {
            //ASSERT(nWritten == nBytes);
            break;
        }
    }
}

void CWaveOut::Wait()
{
    /*  Send any remaining buffers */
    Flush();

    /*  Wait for our buffers back */
    for (int i = 0; i < m_nBuffers; i++) {
        WaitForSingleObject(m_hSem, INFINITE);
    }
    LONG lPrevCount;
    ReleaseSemaphore(m_hSem, m_nBuffers, &lPrevCount);
}

/**************************************************************************

  End of wave player stuff

 **************************************************************************/


HRESULT RenderStreamToDevice(IMultiMediaStream *pMMStream)
{
    WAVEFORMATEX wfx;
    #define DATA_SIZE 5000
    PBYTE pBuffer = (PBYTE)LocalAlloc(LMEM_FIXED, DATA_SIZE);

    IMediaStream *pStream;
    IAudioStreamSample *pSample;
    IAudioMediaStream *pAudioStream;
    IAudioData *pAudioData;

    pMMStream->GetMediaStream(MSPID_PrimaryAudio, &pStream);
    pStream->QueryInterface(IID_IAudioMediaStream, (void **)&pAudioStream);
    pAudioStream->GetFormat(&wfx);
    CoCreateInstance(CLSID_AMAudioData, NULL, CLSCTX_INPROC_SERVER,
                                IID_IAudioData, (void **)&pAudioData);
    pAudioData->SetBuffer(DATA_SIZE, pBuffer, 0);
    pAudioData->SetFormat(&wfx);
    pAudioStream->CreateSample(pAudioData, 0, &pSample);
    HANDLE hEvent = CreateEvent(FALSE, NULL, NULL, FALSE);
    CWaveOut WaveOut(&wfx, 4, 2048);
    int iTimes;
    for (iTimes = 0; iTimes < 3; iTimes++) {
        DWORD dwStart = timeGetTime();
        for (; ; ) {
            HRESULT hr = pSample->Update(0, hEvent, NULL, 0);
            if (FAILED(hr) || MS_S_ENDOFSTREAM == hr) {
                break;
            }
            WaitForSingleObject(hEvent, INFINITE);
            DWORD dwTimeDiff = timeGetTime() - dwStart;
            //  We'll get bored after about 10 seconds
            if (dwTimeDiff > 10000) {
                break;
            }
            DWORD dwLength;
            pAudioData->GetInfo(NULL, NULL, &dwLength);
            WaveOut.Write(pBuffer, dwLength);
        }
        pMMStream->Seek(0);
    }

    pAudioData->Release();
    pSample->Release();
    pStream->Release();
    pAudioStream->Release();
    LocalFree((HLOCAL)pBuffer);

    return S_OK;
}

HRESULT RenderFileToMMStream(WCHAR * pszFileName, IMultiMediaStream **ppMMStream)
{
    IAMMultiMediaStream *pAMStream;
    CoCreateInstance(CLSID_AMMultiMediaStream, NULL, CLSCTX_INPROC_SERVER,
                     IID_IAMMultiMediaStream, (void **)&pAMStream);
    pAMStream->Initialize(STREAMTYPE_READ, AMMSF_NOGRAPHTHREAD, NULL);
    pAMStream->AddMediaStream(NULL, &MSPID_PrimaryAudio, 0, NULL);
    pAMStream->OpenFile(pszFileName, AMMSF_RUN);
    *ppMMStream = pAMStream;
    return S_OK;
}

int _CRTAPI1 main(int argc, char *argv[])
{
    IMultiMediaStream *pMMStream;
    CoInitialize(NULL);
    WCHAR wszName[1000];
    MultiByteToWideChar(CP_ACP, 0, argv[1], -1, wszName,
                        sizeof(wszName) / sizeof(wszName[0]));
    RenderFileToMMStream(wszName, &pMMStream);
    RenderStreamToDevice(pMMStream);
    pMMStream->Release();
    CoUninitialize();
    return 0;
}
```

Previous Home Topic Contents Index Next

DirectDrawEx

This section contains an overview of the DirectDrawEx dynamic-link library, which extends the current functionality of Microsoft® DirectDraw®. It also contains reference material for the DirectDrawEx interfaces, IDirectDrawFactory and IDirectDraw3.

- Using DirectDrawEx
- IDirectDrawFactory Interface
- IDirectDraw3 Interface

Using DirectDrawEx

This article provides a brief overview of DirectDrawEx and how it extends the functionality of a DirectDraw object as described in the Microsoft DirectX® SDK.

Contents of this article:

- What Is DirectDrawEx?
- Advantages of Using DirectDrawEx
- Creating DirectDraw Objects and Surfaces with DirectDrawEx
- Distinctions Between DirectDraw and DirectDrawEx

What Is DirectDrawEx?

DirectDrawEx is a dynamic-link library (DLL) that extends current functionality of DirectDraw, enhancing existing features and providing new functionality. DirectDrawEx also exposes new interfaces that applications can use when you include the Ddrawex.h header file.

To create a DirectDraw object that can use the extended features provided by DirectDrawEx, you must create the object by using the IDirectDrawFactory interface. A DirectDraw object created with the **IDirectDrawFactory** interface will support the IDirectDraw3 interface, aggregation of DirectDraw surfaces, data exchange, and palette mapping, in addition to the features of DirectDraw objects described in the DirectX SDK.

Advantages of Using DirectDrawEx

The primary advantage of creating a DirectDraw object through the IDirectDrawFactory interface is that it exposes the IDirectDraw3 interface. The **IDirectDraw3** interface inherits all the functionality of the IDirectDraw and the IDirectDraw2 interfaces and provides a new method that can retrieve a pointer to an IDirectDrawSurface interface, given a handle to a device context.

To obtain the IDirectDraw3 interface, you must call the IDirectDrawFactory::CreateDirectDraw method to create the DirectDraw object and expose the IUnknown and IDirectDraw interfaces. Applications can then call QueryInterface to obtain a pointer to the **IDirectDraw3** interface. To view sample code that demonstrates this, see Creating DirectDraw Objects and Surfaces with DirectDrawEx.

Another advantage of using DirectDrawEx over using DirectDraw is that you can now aggregate inner objects with outer objects by using the IDirectDraw3::CreateSurface method. Formerly, IDirectDraw::CreateSurface and IDirectDraw2::CreateSurface did not provide COM aggregation features. For a thorough description of how IDirectDraw3 implements aggregation see, **IDirectDraw3::CreateSurface**.

Finally, DirectDrawEx now also provides the DDSCAPS_DATAEXCHANGE flag for the DDSCAPS structure's **dwcaps** member. When a surface is created using the DDSCAPS_DATAEXCHANGE flag, the surface will automatically be moved into video memory if there is enough video memory available, otherwise a system memory surface will be created. This stabilizes video memory surfaces and ensures that they will not be lost, even if system memory decides to move them into video memory in the future.

Creating DirectDraw Objects and Surfaces with DirectDrawEx

The following sample code demonstrates how to create a DirectDraw object by using DirectDrawEx, and get a pointer to the IDirectDraw3 interface. The code shows how to create and call DirectDraw objects.

```
#include ddrawex.h

void CreateDDEx()
{
        //Declarations
        HRESULT         hr;
        IDirectDraw     *pDD;
        IDirectDraw3    *pDD3;
        IDirectDrawFactory *pDDF;

        //Initialize COM library
        CoInitialize(NULL);


        //Create a DirectDrawFactory object and get
        //an IDirectDrawFactory interface pointer.
        CoCreateInstance(CLSID_DirectDrawFactory, NULL, CLSCTX_INPROC_SERVER,
                                                  IID_IDirectDrawFactory, (vc

        //Call the IDirectDrawFactory::CreateDirectDraw method to create the
        //DirectDraw surface, set the cooperative level, and get the address
        //of an IDirectDraw interface pointer.
        hr = (pDDF->CreateDirectDraw(NULL, GetDesktopWindow(), DDSCL_NORMAL,
                                NULL, NULL, &pDD));
        if (hr !=DD_OK) {//error checking
        }

        //Now query for the new IDirectDraw3 interface, and release the old one.
        hr =(pDD->QueryInterface(IID_IDirectDraw3, (LPVOID*)&pDD3));

        if (hr !=S_OK) {//error checking
        }

        //Release IDirectDraw.
        pDD->Release();
        pDD= NULL;

        //Initialize the DDSURFACEDESC structure for the primary surface.
        ZeroMemory(&ddsd, sizeof(ddsd));
      ddsd.dwSize = sizeof(ddsd);
        ddsd.dwFlags = DDSD_CAPS;
      ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;
      hr = pDD3->CreateSurface(&ddsd, &pPrimarySurface, NULL);


        //Do whatever you need to do in your application here with your
        //DirectDraw surface.

        //Release IDirectDraw3, IDirectDrawFactory, and the DirectDraw surface.
        pDD3->Release();
        pDDF->Release();
        pPrimarySurface->Release();

        //Close the COM library
        CoUninitialize();
}
```

Distinctions Between DirectDraw and DirectDrawEx

One important distinction to note between DirectDrawEx and DirectDraw is that applications that have created multiple DirectDrawSurface objects through a DirectDrawEx surface must release every DirectDraw surface.

Also, calling the GetDDInterface method from any surface created under DirectDrawEx will return a pointer to the IUnknown interface instead of a pointer to an IDirectDraw interface. Applications must use the IUnknown::QueryInterface method to retrieve the **IDirectDraw**, IDirectDraw2, or IDirectDraw3 interfaces.

Finally, DirectDrawEx does not currently support blitting between surfaces created by DirectDrawEx and surfaces created by DirectDraw. Applications should blit only between similar surfaces.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawFactory Interface

The **IDirectDrawFactory** interface is used to create and enumerate DirectDraw objects that support the extended features of DirectDrawEx (see Using DirectDrawEx for more information).

When to Implement

Do not implement this interface; DirectDrawEx implements it for you.

When to Use

Use this interface in an application when you want to create a DirectDrawEx object.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDirectDrawFactory methods	**Description**
CreateDirectDraw	Creates a DirectDraw object and retrieves pointers to the IUnknown and the IDirectDraw interfaces.
DirectDrawEnumerate	Enumerates the DirectDraw surfaces installed on the system.

Previous Home Topic Contents Index Next

IDirectDrawFactory::CreateDirectDraw

IDirectDrawFactory Interface

Creates a DirectDraw object and retrieves pointers to the IUnknown and IDirectDraw interfaces.

```
STDMETHOD CreateDirectDraw(
  GUID * pGUID,
  HWND hWnd,
  DWORD dwCoopLevelFlags,
  DWORD dwReserved,
  IUnknown *pUnkOuter,
  IDirectDraw **ppDirectDraw
  ) PURE;
```

Parameters

pGUID

[out] Pointer to the globally unique identifier (GUID) that represents the driver to be created. Set this to NULL to indicate the active display driver, or you can pass one of the following flags to restrict the active display driver's behavior for debugging purposes:

Value	Meaning
DDCREATE_EMULATIONONLY	The DirectDraw object will use emulation for all features; it will not take advantage of any hardware-supported features.
DDCREATE_HARDWAREONLY	The DirectDraw object will never emulate features not supported by the hardware. Attempts to call methods that require unsupported features will fail, returning DDERR_UNSUPPORTED (operation not supported).

hWnd

[in] Window handle to the application.

dwCoopLevelFlags

[in] Application's top-level behavior. Specify one or more of the following flags:

Value	Meaning
DDSCL_ALLOWMODEX	Enables the use of Mode X display modes. You must use this flag with the DDSCL_EXCLUSIVE and DDSCL_FULLSCREEN flags.
DDSCL_ALLOWREBOOT	Enables the user to reboot by pressing CTRL+ALT+DEL while the application is in full-screen exclusive mode.
DDSCL_EXCLUSIVE	Requests the exclusive level. You must use this flag with the DDSCL_FULLSCREEN flag. At the full screen and exclusive cooperative level, you can use the hardware to its fullest. In this mode, you can set custom and dynamic palettes, change display resolutions, compact memory, and implement page flipping. The exclusive (full-screen) mode does not prevent other applications from allocating surfaces, nor does it exclude them from using DirectDraw or GDI. However, it does prevent applications other than the one currently with exclusive access from changing the display mode or palette.
DDSCL_FULLSCREEN	Indicates that the exclusive-mode owner will be in control of the entire primary surface. You must use this flag with the DDSCL_EXCLUSIVE flag.
DDSCL_NORMAL	Indicates that the application will function as a regular Windows® application. You can't use this flag with the DDSCL_ALLOWMODEX, DDSCL_EXCLUSIVE, or DDSCL_FULLSCREEN flags.
DDSCL_NOWINDOWCHANGES	Indicates that DirectDraw can't minimize or restore the application window when the application is activated.

dwReserved

[in] Reserved for future use. Must be NULL.

pUnkOuter

[in] Pointer to an IUnknown interface on an outer object that will be aggregated with an inner object's **IUnknown** interface.

ppDirectDraw

[out] Address of a pointer to an IDirectDraw interface.

Return Values

Returns DD_OK if successful, or one of the following error values otherwise:

Value	Meaning
E_OUTOFMEMORY	There isn't enough memory available to create a DirectDraw object.
DDERR_GENERIC	There is an undefined error condition.
DDERR_UNSUPPORTED	DirectDraw doesn't support the operation.
DDERR_DIRECTDRAWALREADYCREATED	A DirectDrawEx object representing this driver has already been created for this process.
DDERR_INVALIDDIRECTDRAWGUID	The GUID passed to this method is not a valid DirectDrawEx driver identifier.
DDERR_INVALIDPARAMS	One or more of the parameters passed to the method are incorrect.
DDERR_NODIRECTDRAWHW	Hardware-only DirectDrawEx object creation isn't possible; the driver doesn't support any hardware.

Remarks

This method creates DirectDraw objects in the same way that the DirectDrawCreate function is used to create DirectDraw objects, and sets cooperative levels the same way the IDirectDraw2::SetCooperativeLevel method sets cooperative levels. However, in addition to creating a DirectDraw object, successful calls to the **IDirectDrawFactory::CreateDirectDraw** method will obtain a pointer to the IUnknown and IDirectDraw interfaces, which are exposed on the DirectDraw object. Applications can now query the DirectDraw object to obtain the address of a pointer to an IDirectDraw3 interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawFactory::DirectDrawEnumerate

IDirectDrawFactory Interface

Enumerates the DirectDraw objects installed on the system.

STDMETHOD DirectDrawEnumerate(
  LPDDENUMCALLBACK *lpCallback*,
  LPVOID *lpContext*
) PURE;

Parameters

lpCallback
: [in] Pointer to a DDEnumCallback function that will be called with a description of each DirectDrawEx-enabled hardware abstraction layer (HAL) installed in the system.

lpContext
: [in] Address of an application-defined structure that the system will pass to the callback function each time the function is called.

Return Values

Returns DD_OK if successful, or DDERR_INVALIDPARAMS otherwise.

Remarks

This method functions in a similar manner to the **DirectDrawEnumerate** function, defined in the DirectX SDK.

Your application can call this method only after a successful call to the IDirectDrawFactory::CreateDirectDraw method.

Previous Home Topic Contents Index Next

IDirectDraw3 Interface

The **IDirectDraw3** interface is available to applications that have created a DirectDraw object through the IDirectDrawFactory::CreateDirectDraw method. This method retrieves the address of a pointer to an IDirectDraw interface, which your application can use to query for the **IDirectDraw3** interface.

The **IDirectDraw3** interface extends the IDirectDraw and the IDirectDraw2 interfaces by adding the IDirectDraw3::GetSurfaceFromDC method and providing new behavior for the IDirectDraw3::CreateSurface method (formerly IDirectDraw2::CreateSurface). This section provides information on the new behavior of the CreateSurface method. See the DirectX SDK for information on the original behavior of this and all other **IDirectDraw** and **IDirectDraw2** methods as they relate to DirectDraw objects.

For more information on additional functionality provided by DirectDrawEx, see Using DirectDrawEx.

When to Implement

Do not implement this interface; DirectDrawEx implements it for you.

When to Use

Applications use this interface when they have created a DirectDraw object through a successful call to the IDirectDrawFactory::CreateDirectDraw method. Applications can use this interface to get an IDirectDrawSurface interface pointer directly from a handle to a device context.

Methods in Vtable Order

IUnknown methods	**Description**
QueryInterface	Retrieves pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IDirectDraw3 methods	**Description**
CreateSurface	Creates a DirectDrawSurface object from a DirectDraw object.
GetSurfaceFromDC	Retrieves a pointer to an IDirectDrawSurface interface from a handle to a device context.

IDirectDraw3::CreateSurface

IDirectDraw3 Interface

Creates a DirectDrawSurface object from a DirectDraw object.

STDMETHOD CreateSurface(
  LPDDSURFACEDESC *lpDDSurfaceDesc*,
  LPDIRECTDRAWSURFACE FAR **lpDDSurface*,
  IUnknown FAR **pUnkOuter*
) PURE;

Parameters

lpDDSurfaceDesc
: [in] Pointer to the DDSURFACEDESC structure that describes the requested surface. You should set any unused members of **DDSURFACEDESC** to zero before calling this method. A DDSCAPS structure is a member of **DDSURFACEDESC**.

lpDDSurface
: [out] Address of a pointer to be initialized with a valid DirectDrawSurface pointer if the call succeeds.

pUnkOuter
: [in] Pointer to an IUnknown interface on an outer object that will be aggregated with an inner object's **IUnknown** interface.

Return Values

Returns DD_OK if successful, or one of the following error values otherwise:

Value	Meaning
DDERR_INCOMPATIBLEPRIMARY	The primary surface creation request does not match the existing primary surface.
DDERR_INVALIDCAPS	One or more of the capability bits passed to the callback function are incorrect.
DDERR_INVALIDOBJECT	DirectDraw received a pointer to an invalid DirectDraw object.
DDERR_INVALIDPARAMS	One or more of the parameters passed to the method are incorrect.
DDERR_INVALIDPIXELFORMAT	The pixel format was invalid as specified.
DDERR_NOALPHAHW	No alpha acceleration hardware is present or available, which caused the requested operation to fail.
DDERR_NOCOOPERATIVELEVELSET	The IDirectDraw2::SetCooperativeLevel method was not called before the surface was created.
DDERR_NODIRECTDRAWHW	Hardware-only DirectDraw object creation isn't possible; the driver doesn't support any hardware.
DDERR_NOEMULATION	Software emulation isn't available.
DDERR_NOEXCLUSIVEMODE	The operation requires the application to have exclusive mode, but the application doesn't have exclusive mode.
DDERR_NOFLIPHW	Flipping visible surfaces isn't supported.
DDERR_NOMIPMAPHW	The operation can't be carried out because no mipmap texture mapping hardware is present or available.
DDERR_NOOVERLAYHW	The operation can't be carried out because no overlay hardware is present or available.
DDERR_NOZBUFFERHW	The operation to create a z-buffer in display memory or to perform a blit using a z-buffer can't be carried out because there is no hardware support for z-buffers.
DDERR_OUTOFMEMORY	DirectDraw doesn't have enough available memory to perform the operation.
DDERR_OUTOFVIDEOMEMORY	DirectDraw doesn't have enough display memory to perform the operation.
DDERR_PRIMARYSURFACEALREADYEXISTS	The application has already created a primary surface.
DDERR_UNSUPPORTEDMODE	The operation isn't supported.

Remarks

Passing in NULL for the *pUnkOuter* parameter will return the address of a DirectDraw surface in the *lpDDSurface* parameter. However, if you pass in a pointer to an outer interface you want to aggregate with an inner interface, you will get back an IUnknown pointer for the *lpDDSurface* parameter.

DirectDrawEx now also provides the DDSCAPS_DATAEXCHANGE flag for the *dwcaps* member of the DDSCAPS structure, which is defined as a combination of DDSCAPS_SYSTEMMEMORY and

DDSCAPS_VIDEOMEMORY in Ddrawex.h. When a surface is created using the DDSCAPS_DATAEXCHANGE flag, the surface will be automatically moved into video memory if there is enough video memory available; otherwise, a system memory surface will be created. Also, setting this flag in conjunction with the DDSCAPS_OWNDC flag enables applications to call the IDirectDrawSurface::GetDC method to lock the device context for as long they require, without holding a lock on the surface.

This method calls the IDirectDraw::CreateSurface and IDirectDraw2::CreateSurface methods.

IDirectDraw3::GetSurfaceFromDC

IDirectDraw3 Interface

Retrieves a pointer to an IDirectDrawSurface interface from a handle to a device context.

STDMETHOD GetSurfaceFromDC(
  HDC *hdc*,
  IDirectDrawSurface ***ppSurface*
) PURE;

Parameters

hdc
  [in] Handle of the device context (DC).
ppSurface
  [out] Address of a pointer to an IDirectDrawSurface interface.

Return Values

Returns S_OK if successful, or one of the following values otherwise:

Value	Meaning
E_POINTER	Invalid pointer to IDirectDrawSurface.
DDERR_NOTFOUND	The requested item wasn't found.

Appendixes

This section contains lists of the media types supported by DirectShow, the various MPEG media types, reserved identifiers in DirectShow, DVD video formats, and further reading.

- Media Types
- MPEG-1 Media Types
- Time Stamps
- Sample Properties
- CLSIDs in DirectShow
- DirectShow DVD Support
- Country Codes and Channel to Frequency Mappings
- Reserved Identifiers
- Further Reading

Media Types

Microsoft® DirectShow™ uses the AM_MEDIA_TYPE structure to describe media samples. This structure includes GUID fields for major type, subtype, and format type, as well as fields specifying other sample features, such as whether the samples are compressed. This article summarizes the major type and subtype options registered by DirectShow. These media types are defined in Uuids.h.

- Media Types with No Subtype
- Audio Media Types
- Line21 Media Types

- MPEG2 Media Types
- Stream Media Types
- Video Media Types
- Analog Video Media Types

Media Types with No Subtype

The following table describes the media types with no subtype.

MEDIATYPEs with no MEDIASUBTYPEs	
MEDIATYPE_AnalogAudio	Analog audio connection
MEDIATYPE_File	Media type is a file, used closed captions.
MEDIATYPE_Interleaved	Data is interleaved, used by Digital Video (DV).
MEDIATYPE_Midi	Data is MIDI format.
MEDIATYPE_ScriptCommand	Data is a script command, used by closed captions.
MEDIATYPE_Text	Data is text.
MEDIATYPE_Timecode	Data is timecode data.

Audio Media Types

The **wFormatTag** field in the WAVEFORMATEX structure specifies the audio format type. The format type is generally FORMAT_WaveFormatEx. Media samples are generally whole number of samples as specified in the **wBitsPerSample** field in the WAVEFORMATEX structure. This is not necessarily true for MPEG audio samples that can come from packetized streams and are therefore not necessarily packaged on sample / frame boundaries. For MPEG audio the time stamp in a media sample is the time stamp for the first frame whose first byte is contained in the media sample.

Media subtypes are defined for each **wFormatTag** as follows:

- The Data1 subfield of the Media Subtype is the same as the **wFormatTag** value.
- The Data 2 field is 0.
- The Data 3 field is 0x0010.
- The Data 4 field is 0x80, 0x00, 0x00, 0xAA, 0x00, 0x38, 0x9B, 0x71.

Thus, for PCM audio the subtype GUID would be:

```
{00000001-0000-0010-8000-00AA00389B71}
```

Older filters may still use GUID_NULL as the subtype so this should be checked for. However, registration of a filter with the explicit subtype greatly improves the speed of graph loading, especially when the given filter is not required. The CreateAudioMediaType function supplied in the DirectShow SDK can be used to create an AM_MEDIA_TYPE structure from a WAVEFORMATEX Structure.

The following table describes the audio media subtypes.

MEDIATYPE_Audio	**Data is audio**
MEDIASUBTYPE_PCMAudio	PCM audio
MEDIASUBTYPE_MPEG1Packet	MPEG1 Audio packet
MEDIASUBTYPE_MPEG1Payload	MPEG1 Audio Payload

Line21 Media Types

The following table describes the Line21 closed captioning media subtypes.

MEDIATYPE_AUXLine21Data	**Data is Line21 type, used by closed captions**
MEDIASUBTYPE_Line21_BytePair	Line21 data as byte pairs
MEDIASUBTYPE_Line21_GOPPacket	Line21 data in DVD GOP Packet
MEDIASUBTYPE_Line21_VBIRawData	Line21 data in raw VBI format

MPEG2 Media Types

The following table describes the MPEG2 media subtypes.

MEDIATYPE_MPEG2_PES	**Data is MPEG2 format, used by DVD**
MEDIASUBTYPE_DVD_SUBPICTURE	Subpicture data
MEDIASUBTYPE_DOLBY_AC3	Dolby data
MEDIASUBTYPE_MPEG2_AUDIO	MPEG2 audio data
MEDIASUBTYPE_DVD_LPCM_AUDIO	DVD audio data

Stream Media Types

Time stamps are byte positions * 10000000 (notionally 1 byte per second) rather than real times.

The following table describes the stream media subtypes.

MEDIATYPE_Stream	**Data is a non-timestamped byte stream**
MEDIASUBTYPE_Avi	Data from AVI file
MEDIASUBTYPE_WAVE	Data from WAV file
MEDIASUBTYPE_AU	Data from AU file
MEDIASUBTYPE_AIFF	Data from AIFF file
MEDIASUBTYPE_MPEG1Video	MPEG video
MEDIASUBTYPE_MPEG1System	MPEG system
MEDIASUBTYPE_MPEG1VideoCD	MPEG video CD
MEDIASUBTYPE_MPEG1Audio	MPEG audio
MEDIASUBTYPE_DssVideo	Dss Video
MEDIASUBTYPE_DssAudio	Dss Audio

Video Media Types

The following table describes the video media subtypes.

MEDIATYPE_Video	**Data is video**
MEDIASUBTYPE_YVU9	Standard YVU9 format uncompressed data. A planar YUV format. A Y sample at every pixel, a U and V sample at every fourth pixel horizontally on each line; a Y sample on every vertical line, a U and V sample at every fourth vertical line. 9 bits per pixel.
MEDIASUBTYPE_Y411	YUV 411 format data. Same as Y41P.