IAMFileCutListElement::GetTrimInPosition

IAMFileCutListElement Interface

Retrieves the media time of the trimin point, based on the timeline of the cut's source file.

HRESULT GetTrimInPosition(
REFERENCE_TIME **pmtTrimIn*
);

Parameters

pmtTrimIn
[out] Pointer that will receive the trimin point.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_POINTER	Null pointer argument.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMFileCutListElement::GetTrimLength

IAMFileCutListElement Interface

Retrieves the length of time between the trimin and trimout points.

HRESULT GetTrimLength(
REFERENCE_TIME **pmtLength*
);

Parameters

pmtLength
[out] Pointer that will receive the length in media time.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_POINTER	Null pointer argument.
S_OK	Success.

Remarks

This method retrieves the length of time between the in and out points specified by GetTrimInPosition and GetTrimOutPosition.

The value that **GetTrimLength** retrieves equals the value that GetElementDuration retrieves (trimout minus trimin). Other lengths are not supported.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMFileCutListElement::GetTrimOutPosition

IAMFileCutListElement Interface

Retrieves the media time of the trimout point, based on the timeline of the cut's source file.

HRESULT GetTrimOutPosition(
REFERENCE_TIME **pmtTrimOut*
);

Parameters

pmtTrimOut
[out] Pointer that will receive the trimout point, in REFERENCE_TIME.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_POINTER	Null pointer argument.
S_OK	Success.

Remarks

The media time does not include the trimout point.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMLine21Decoder Interface

The **IAMLine21Decoder** interface provides access to closed-captioned information and settings. Closed-captioned information is transmitted in the vertical blanking interval (VBI) of television signals, specifically on line 21 (Line21) of field 1 in the VBI. Video cassette recorders record this information on video tape, and you can use Microsoft® DirectShow™ filters to capture the Line21 data and save it on disk in a media file format such as audio-video interleaved (AVI). The closed-captioned information appears as a separate stream within the media file.

Closed-captioned text is currently used mainly in digital versatile disc (DVD) movies. DVD movies contain Line21 data as part of the user data section of each Group of Pictures (GOP) in the video stream. Capture cards with Windows Driver Model (WDM) drivers will provide Line21 data.

When to Implement

Do not implement this interface. DirectShow provides the Line 21 Decoder in the DirectX Media 5.1 SDK, which implements it for you.

When to Use

Applications use this interface when they want to provide closed-captioned text, primarily to turn closed-captioned capabilities on and off. Use this interface in your application or in the filter immediately downstream of the Line21 Decoder filter (typically a mixer filter) to change closed-captioned options, such as the output video's size and whether to make the caption background opaque or transparent. Mixer filters can also change the physical color used for the

background color key.

Applications can call the GetDrawBackgroundMode and SetDrawBackgroundMode methods so the user can select transparent or opaque captioning.

Methods in Vtable Order

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

IAMLine21Decoder methods	Description
GetDecoderLevel	Retrieves the closed-captioned decoder level.
GetCurrentService	Retrieves the current closed captioning service selected by the user.
SetCurrentService	Sets the current closed captioning service.
GetServiceState	Retrieves the closed captioning service state (on or off).
SetServiceState	Sets the closed captioning service state.
GetOutputFormat	Retrieves information about output video characteristics such as size and bit depth.
SetOutputFormat	Sets information that describes output video characteristics such as size and bit depth.
GetBackgroundColor	Retrieves the physical color to use as background for overlays.
SetBackgroundColor	Sets the physical color to use as background for overlays.
GetRedrawAlways	Retrieves whether the renderer should redraw the whole output bitmap for each sample.
SetRedrawAlways	Sets whether the renderer should redraw the whole output bitmap for each sample.
GetDrawBackgroundMode	Retrieves whether the caption text background should be opaque or transparent.
SetDrawBackgroundMode	Sets whether to make the caption text background opaque or transparent.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetBackgroundColor

IAMLine21Decoder Interface

Retrieves the physical color to use as background for overlays.

HRESULT GetBackgroundColor(
DWORD **pdwPhysColor*
);

Parameters

pdwPhysColor
Pointer to the retrieved DWORD value.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

Magenta is the default background color.

See Also

IAMLine21Decoder::SetBackgroundColor

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetCurrentService

IAMLine21Decoder Interface

Retrieves the current closed captioning service selected by the user.

HRESULT GetCurrentService(
AM_LINE21_CCSERVICE **lpService*
);

Parameters

lpService
Pointer to the current service. This value is a member of the AM_LINE21_CCSERVICE enumerated data type. The default service is AM_L21_CCSERVICE_Caption1.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

See Also

IAMLine21Decoder::SetCurrentService

IAMLine21Decoder::GetDecoderLevel

IAMLine21Decoder Interface

Retrieves the closed-captioned decoder level.

```
HRESULT GetDecoderLevel(
  AM_LINE21_CCLEVEL *lpLevel
  );
```

Parameters

lpLevel
Pointer to the retrieved decoder level. AM_L21_CCLEVEL_TC2 (TC2) is the only supported operating channel level and is an enhanced and backward-compatible version of the original TC1 level.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

This method is for informational purposes only.

TC1 and TC2 are television set decoder levels that represent whether the television can handle some closed-captioned byte pairs and produce the desired captioning results. The Line21 Decoder is capable of TC2 level decoding, which includes all TC1 decoding. Only the first 100,000 television sets manufactured that included closed-captioned capability were TC1 compliant; the later TV sets are TC2 compliant.

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetDrawBackgroundMode

IAMLine21Decoder Interface

Retrieves whether the caption text background should be opaque or transparent.

HRESULT GetDrawBackgroundMode(
AM_LINE21_DRAWBGMODE **lpMode*
);

Parameters

lpMode
Retrieved mode. Supported mode values are AM_L21_DRAWBGMODE_Opaque and AM_L21_DRAWBGMODE_Transparent.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

By default, the caption background is opaque.

See Also

IAMLine21Decoder::SetDrawBackgroundMode

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetOutputFormat

IAMLine21Decoder Interface

Retrieves information about output video characteristics such as size and bit depth.

HRESULT GetOutputFormat(
LPBITMAPINFOHEADER *lpbmih*
);

Parameters

lpbmih
Pointer to the retrieved BITMAPINFOHEADER structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

If successful, the default implementation returns S_FALSE if downstream filters haven't defined an output format, or S_OK if an output format has been defined.

Remarks

The default video output size is 320 × 240 pixels.

See Also

IAMLine21Decoder::SetOutputFormat

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetRedrawAlways

IAMLine21Decoder Interface

Retrieves whether the renderer should redraw the whole output bitmap for each sample.

HRESULT GetRedrawAlways(
LPBOOL *lpbOption*
);

Parameters

lpbOption
Pointer to a value indicating whether the whole bitmap should be redrawn; FALSE by default, indicating don't always redraw. TRUE means always redraw.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

See Also

IAMLine21Decoder::SetRedrawAlways

Previous Home Topic Contents Index Next

IAMLine21Decoder::GetServiceState

IAMLine21Decoder Interface

Retrieves the closed captioning service state (on or off).

```
HRESULT GetServiceState(
  AM_LINE21_CCSTATE *lpState
  );
```

Parameters

lpState
Pointer to the retrieved state. Supported state values are AM_L21_CCSTATE_On and AM_L21_CCSTATE_Off. Closed-captioned text is off by default.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

See Also

IAMLine21Decoder::SetServiceState

Previous Home Topic Contents Index Next

IAMLine21Decoder::SetBackgroundColor

IAMLine21Decoder Interface

Sets the physical color to use as background for overlays.

HRESULT SetBackgroundColor(
DWORD *dwPhysColor*
);

Parameters

dwPhysColor
DWORD value that specifies the physical background color.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

Magenta is the default background color.

See Also

IAMLine21Decoder::GetBackgroundColor

Previous Home Topic Contents Index Next

IAMLine21Decoder::SetCurrentService

IAMLine21Decoder Interface

Sets the current closed captioning service.

HRESULT SetCurrentService(
AM_LINE21_CCSERVICE *Service*
);

Parameters

Service
Specified service. This value is a member of the AM_LINE21_CCSERVICE enumerated data type. The default service is AM_L21_CCSERVICE_Caption1.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

See Also

IAMLine21Decoder::GetCurrentService

Previous Home Topic Contents Index Next

IAMLine21Decoder::SetDrawBackgroundMode

IAMLine21Decoder Interface

Sets whether to make the caption text background opaque or transparent.

```
HRESULT SetDrawBackgroundMode(
  AM_LINE21_DRAWBGMODE Mode
  );
```

Parameters

Mode
Mode to set. Supported mode values are AM_L21_DRAWBGMODE_Opaque and AM_L21_DRAWBGMODE_Transparent.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

By default, the caption background is opaque.

See Also

IAMLine21Decoder::GetDrawBackgroundMode

IAMLine21Decoder::SetOutputFormat

IAMLine21Decoder Interface

Sets information that describes output video characteristics such as size and bit depth.

HRESULT SetOutputFormat(
LPBITMAPINFO *lpbmi*
);

Parameters

lpbmi
Pointer to the specified BITMAPINFO structure containing the desired output format.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

The default video output size is 320 × 240 pixels.

See Also

IAMLine21Decoder::GetOutputFormat

IAMLine21Decoder::SetRedrawAlways

IAMLine21Decoder Interface

Sets whether the renderer should redraw the whole output bitmap for each sample.

HRESULT SetRedrawAlways(
BOOL *bOption*
);

Parameters

bOption
Value indicating whether the whole bitmap should be redrawn. TRUE indicates redraw always, FALSE means do not redraw always.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

Remarks

Call this method from your filter if it dirties the buffer that it provides to the Line21 Decoder filter. Typically, a mixer filter resides in the filter graph directly downstream from the Line21 Decoder filter. The mixer filter should call this method and set *bOption* to TRUE to ensure the entire bitmap is redrawn properly.

A downstream mixer (or any filter that needs to do so) should only call this method with *bOption* set to TRUE if it provides the same buffer to the Line21 decoder as it uses to mix secondary video streams(s).

Redrawing (setting *bOption* to TRUE) degrades performance and increases CPU load, because it negates any potential optimizations.

See Also

IAMLine21Decoder::GetRedrawAlways

Previous Home Topic Contents Index Next

IAMLine21Decoder::SetServiceState

IAMLine21Decoder Interface

Sets the closed captioning service state.

HRESULT SetServiceState(
AM_LINE21_CCSTATE *State*
);

Parameters

State
Specified state. Supported state values are AM_L21_CCSTATE_On and AM_L21_CCSTATE_Off. Closed-captioned text is off by default.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if a parameter is invalid or NOERROR to indicate success.

See Also

IAMLine21Decoder::GetServiceState

IAMovie Interface

The **IAMovie** interface is a plug-in distributor (PID) interface that you could use as a replacement for all other interfaces on the filter graph manager. It wraps the most commonly used methods of IGraphBuilder, IMediaControl, IMediaEvent, and IMediaPosition by providing the same methods as these interfaces and simply calling these interfaces on the filter graph manager for the implementation. It also adds several unique methods that simplify instantiating and running a filter graph.

When to Implement

This interface is implemented by the IAMovie sample plug-in distributor included in this SDK. It is not expected that anything else will implement this interface.

When to Use

Applications use plug-in distributors. If this interface is implemented through the IAMovie sample application, you can use the methods on this interface rather than the methods on the IGraphBuilder, IMediaControl, IMediaEvent, and IMediaPosition interfaces. You can also use specialized methods on this interface to render a filter graph and play it in one command, to enumerate filters in the filter graph that contain a specified interface, to enumerate all pins in the filter graph, and to perform other tasks.

Methods in Vtable Order

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

IFilterGraph methods	Description
AddFilter	Adds a filter to the graph and gives it a name.
RemoveFilter	Removes a filter from the graph.
Reconnect	Breaks the existing pin connection and reconnects it to the same pin.
EnumFilters	Provides an enumerator for all filters in the graph.
FindFilterByName	Finds a filter that was added with a specified name.
ConnectDirect	Connects the two IPin objects directly (without intervening filters).
Reconnect	Breaks the existing pin connection and reconnects it to the same pin.
Disconnect	Disconnects the pin, if connected.
SetDefaultSyncSource	Sets the default synchronization source (a clock).

IAMovie methods	Description
Connect	Connects two IPin objects. If they will not connect directly, this method connects them with intervening transforms.
Render	Adds a chain of filters to this output pin so as to render it.
Run	Switches the entire filter graph into running mode.
Pause	Pauses all filters in the filter graph.
Stop	Switches all filters in the filter graph to a stopped state.
GetState	Retrieves the state of the filter graph.
RenderFile	Adds and connects filters needed to play the specified file.
AddSourceFilter	Adds to the graph the source filter that can read the given file name, and returns an interface pointer to the filter object.
GetEventHandle	Retrieves a handle to a manual-reset event that will be signaled.
GetEvent	Retrieves the next notification event.
WaitForCompletion	Waits until the graph's operation has completed.
CancelDefaultHandling	Cancels any default handling of the specified event by the filter graph.
RestoreDefaultHandling	Restores default handling for this event.
get_Duration	Retrieves the total duration of the media stream.
put_CurrentPosition	Sets the time that the media stream begins.
get_CurrentPosition	Retrieves the current position in terms of the total length of the media stream.
get_StopTime	Retrieves the position within the media stream at which playback should stop.
put_StopTime	Sets the position within the media stream at which playback should stop.
get_PrerollTime	Retrieves the time prior to the start position that the filter graph begins any nonrandom access device rolling.
put_PrerollTime	Sets the time prior to the start position that the filter graph begins any nonrandom access device rolling.
put_Rate	Sets the playback rate, relative to normal playback of the media stream.

get_Rate	Retrieves the playback rate, relative to normal playback of the media stream.
RemoveAllFilters	Removes all filters from the filter graph.
Play	Plays the media in the current filter graph.
PlayFile	Plays the media in a given file.
EnumFiltersByInterface	Retrieves a list of filters supporting a specified interface.
EnumPins	Retrieves a list of pins in the filter graph.
EnumPinsIn	Retrieves a list of input pins in the filter graph.
EnumPinsOut	Retrieves a list of output pins in the filter graph.
RenderAll	Renders all output pins in the filter graph.
RenderNewFile	Renders a filter graph for a file name, possibly reusing existing filters.
FreeEventParams	Frees the resources associated with an event's parameters.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMovie::AddSourceFilter

IAMovie Interface

Adds a source filter to the filter graph for this file. The IGraphBuilder::RenderFile method adds the same source filter.

```
HRESULT AddSourceFilter(
  LPCWSTR lpwstrFileName,
  IBaseFilter* * ppFilter
  );
```

Parameters

lpwstrFileName
[in] Pointer to the file.

ppFilter
[out] Pointer to an IBaseFilter interface on the filter that was added.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IGraphBuilder::AddSourceFilter method. The *lpwstrFileName* file name is used as the filter name when **IGraphBuilder::AddSourceFilter** is called.

IAMovie::CancelDefaultHandling

IAMovie Interface

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

HRESULT CancelDefaultHandling(
long *lEvCode*
);

Parameters

lEvCode
Event code for which to cancel default handling.

Return Values

Returns S_OK if successful, or S_FALSE if the event does not have any default handling.

Remarks

This method simply calls the IMediaEvent::CancelDefaultHandling method.

IAMovie::Connect

IAMovie Interface

Connects the two pins, using intermediates if necessary.

HRESULT Connect(
IPin * *ppinOut*,
IPin * *ppinIn*
);

Parameters

ppinOut
[in] Output pin.
ppinIn
[in] Input pin.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IGraphBuilder::Connect method.

Previous Home Topic Contents Index Next

IAMovie::EnumFiltersByInterface

IAMovie Interface

Retrieves a list of filters supporting a specified interface.

```
HRESULT EnumFiltersByInterface(
  REFIID riid,
  IEnumFilters ** ppEnum
  );
```

Parameters

riid
[in] REFIID of the interface qualifying the search.
ppEnum
[out] Retrieved IEnumFilters enumerator containing the matching filters.

Return Values

Returns an HRESULT value.

Remarks

This method retrieves an IEnumFilters enumerator containing a list of pointers to filters in the filter graph that support a specified interface. Note that the pointers in the list (as returned by IEnumFilters::Next) actually point to the specified *riid* interface on each filter rather than to the IBaseFilter interface. For implementation details of this method, see the IAMovie sample

plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::EnumPins

IAMovie Interface

Retrieves a list of pins in the filter graph.

HRESULT EnumPins(
IEnumPins ** *ppEnum*
);

Parameters

ppEnum
[out] Enumerator containing the list of pins.

Return Values

Returns an HRESULT value.

Remarks

For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::EnumPinsIn

IAMovie Interface

Retrieves a list of input pins in the filter graph.

HRESULT EnumPinsIn(
IEnumPins ** *ppEnum*
);

Parameters

ppEnum
[out] Enumerator containing the list of input pins.

Return Values

Returns an HRESULT value.

Remarks

For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::EnumPinsOut

IAMovie Interface

Retrieves a list of output pins in the filter graph.

```
HRESULT EnumPinsOut(
  IEnumPins ** ppEnum
  );
```

Parameters

ppEnum
[out] Enumerator containing the list of output pins.

Return Values

Returns an HRESULT value.

Remarks

For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::FreeEventParams

IAMovie Interface

Frees the resources associated with an event's parameters.

HRESULT FreeEventParams(
long *lEvCode*,
long *lParam1*,
long *lParam2*
);

Parameters

lEvCode
Event code.
lParam1
Event's first parameter.
lParam2
Event's second parameter.

Return Values

Returns an HRESULT.

Remarks

The *lParam1* and *lParam2* parameters must be LONG values, BSTR values, or IUnknown interface pointers. If an argument is a **LONG** value, **FreeEventParams** does nothing to it. If it is an **IUnknown** interface pointer, its reference count has been incremented. Call its Release method to decrement its reference count after calling **FreeEventParams**. If the argument is a **BSTR** value, free it by calling the task allocator after **FreeEventParams**.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMovie::get_CurrentPosition

IAMovie Interface

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

HRESULT get_CurrentPosition(
REFTIME* *pllTime*
);

Parameters

pllTime
[out] Reference time of the current position.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::get_CurrentPosition method.

‹Previous Home Topic Contents Index Next›

IAMovie::get_Duration

IAMovie Interface

Retrieves the total duration of the media stream.

HRESULT get_Duration(
REFTIME* *plength*
);

Parameters

plength
[out] Returned length of the media stream.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::get_Duration method.

Previous Home Topic Contents Index Next

IAMovie::GetEvent

IAMovie Interface

Retrieves the next notification event.

HRESULT GetEvent(
long * *lEventCode*,
long * *lParam1*,
long * *lParam2*,
long *msTimeout*
);

Parameters

lEventCode
[out] Next event notification.
lParam1
[out] First parameter of the event.
lParam2
[out] Second parameter of the event.
msTimeout
[in] Time, in milliseconds, to wait before assuming that there are no events.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaEvent::GetEvent method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMovie::GetEventHandle

IAMovie Interface

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

HRESULT GetEventHandle(
OAEVENT * *hEvent*
);

Parameters

hEvent
[out] Handle for the event.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaEvent::GetEventHandle method.

Previous Home Topic Contents Index Next

IAMovie::get_PrerollTime

IAMovie Interface

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

HRESULT get_PrerollTime(
REFTIME* *pTime*
);

Parameters

pTime
[out] Returned preroll time.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::get_PrerollTime method.

IAMovie::get_Rate

IAMovie Interface

Retrieves the rate of playback relative to normal playback speed.

HRESULT get_Rate(
double * *pdRate*
);

Parameters

pdRate
[out] Returned rate.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::get_Rate method.

IAMovie::GetState

IAMovie Interface

Determines the state of the filter.

HRESULT GetState(
DWORD *dwMilliSecsTimeout*,
FILTER_STATE * *State*
);

Parameters

dwMilliSecsTimeout
[in] Duration of the time-out, in milliseconds. To block indefinitely, pass INFINITE.
State
[out] Holder of the returned state of the filter. States include stopped, paused, running, or intermediate (in the process of changing).

Return Values

Returns an HRESULT value, which will be VFW_S_STATE_INTERMEDIATE if the state transition is not complete, or S_OK if it has been successfully completed.

Remarks

This method simply calls the IMediaControl::GetState method. Note that the state is returned in a FILTER_STATE structure rather than as an OAFilterState type.

Previous Home Topic Contents Index Next

IAMovie::get_StopTime

IAMovie Interface

Retrieves the time at which the media stream stops.

```
HRESULT get_StopTime(
  REFTIME* pTime
  );
```

Parameters

pTime
[out] Returned stop time.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::get_StopTime method.

Previous Home Topic Contents Index Next

IAMovie::Pause

IAMovie Interface

Pauses all the filters in the filter graph.

HRESULT Pause( );

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaControl::Pause method.

Previous Home Topic Contents Index Next

IAMovie::Play

IAMovie Interface

Plays the media in the current filter graph.

HRESULT Play( );

Return Values

Returns an HRESULT value.

Remarks

This method runs the filter graph to completion by calling IAMovie::Run, and waits for it to complete by calling IAMovie::WaitForCompletion. For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::PlayFile

IAMovie Interface

Plays the media in a given file.

HRESULT PlayFile(
LPCWSTR *strFilename*
);

Parameters

strFilename
[in] Name of the file to play.

Return Values

Returns an HRESULT value.

Remarks

This method calls the IAMovie::RenderNewFile method to build a filter graph capable of rendering the file passed in *strFilename* and then plays the file by calling IAMovie::Play. For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::put_CurrentPosition

IAMovie Interface

Sets the time that the media stream begins.

HRESULT put_CurrentPosition(
REFTIME *Time*
);

Parameters

Time
[in] Start time.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::put_CurrentPosition method.

Previous Home Topic Contents Index Next

IAMovie::put_PrerollTime

IAMovie Interface

Sets the time prior to the start position that devices should start rolling.

HRESULT put_PrerollTime(
REFTIME *llTime*
);

Parameters

llTime
[in] Preroll time to be set.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::put_PrerollTime method.

Previous Home Topic Contents Index Next

IAMovie::put_Rate

IAMovie Interface

Sets the rate of playback relative to normal speed.

HRESULT put_Rate(
double *dRate*
);

Parameters

dRate
[in] Rate to set.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::put_Rate method.

Previous Home Topic Contents Index Next

IAMovie::put_StopTime

IAMovie Interface

Sets the time at which the media stream will stop.

HRESULT put_StopTime(
REFTIME *Time*
);

Parameters

Time
[in] Stop time.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaPosition::put_StopTime method.

Previous Home Topic Contents Index Next

IAMovie::RemoveAllFilters

IAMovie Interface

Removes all filters from the filter graph.

HRESULT RemoveAllFilters( );

Return Values

Returns an HRESULT value.

Remarks

This method enumerates all filters in the filter graph and then removes each of them by calling RemoveFilter. For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::Render

IAMovie Interface

Builds a filter graph that renders the data from this output pin.

HRESULT Render(
IPin * *ppinOut*
);

Parameters

ppinOut
[in] Output pin.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IGraphBuilder::Render method.

Previous Home Topic Contents Index Next

IAMovie::RenderAll

IAMovie Interface

Renders all output pins in the filter graph.

HRESULT RenderAll( );

Return Values

Returns an HRESULT value.

Remarks

This member function enumerates all output pins in the filter graph and renders each of them (builds a filter graph capable of rendering the media type) by calling IAMovie::Render for each output pin. For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::RenderFile

IAMovie Interface

Adds and connects filters needed to play the specified file.

HRESULT RenderFile(
LPCWSTR *strFilename*
);

Parameters

strFilename
Name of the file to render.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IGraphBuilder::RenderFile method with the *PlayList* parameter of that method set to NULL.

Previous Home Topic Contents Index Next

IAMovie::RenderNewFile

IAMovie Interface

Renders a filter graph for a file name, possibly reusing existing filters.

HRESULT RenderNewFile(
LPCWSTR *strFilename*
);

Parameters

strFilename
[in] Name of the file to be rendered.

Return Values

Returns an HRESULT value.

Remarks

This method disconnects all filters in the filter graph, then renders the file in the *strFilename* parameter by calling IAMovie::RenderFile. This will use the disconnected filters if they can be

used to render the file. It then removes any unconnected filters left in the filter graph. For implementation details of this method, see the IAMovie sample plug-in distributor included in this SDK.

Previous Home Topic Contents Index Next

IAMovie::RestoreDefaultHandling

IAMovie Interface

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

HRESULT RestoreDefaultHandling(
long *lEvCode*
);

Parameters

lEvCode
[in] Event to restore.

Return Values

Returns S_OK if successful, or S_FALSE if there is no default handling for this event.

Remarks

This method simply calls the IMediaEvent::RestoreDefaultHandling method.

Previous Home Topic Contents Index Next

IAMovie::Run

IAMovie Interface

Switches the entire filter graph into a running state.

HRESULT Run( );

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaControl::Run method.

Previous Home Topic Contents Index Next

IAMovie::Stop

IAMovie Interface

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

HRESULT Stop( );

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaControl::Stop method.

Previous Home Topic Contents Index Next

IAMovie::WaitForCompletion

IAMovie Interface

Provides a simplified way for applications to wait until the filter graph's operation has completed. It is the equivalent of blocking until the event notification EC_COMPLETE, EC_ERRORABORT, or EC_USERABORT is received.

HRESULT WaitForCompletion(
  long *msTimeout*,

long * *pEvCode*
);

Parameters

msTimeout
[in] Duration of the time-out, in milliseconds. To block indefinitely, pass INFINITE.
pEvCode
[out] Event to wait for.

Return Values

Returns an HRESULT value.

Remarks

This method simply calls the IMediaEvent::WaitForCompletion method.

Previous Home Topic Contents Index Next

IAMovieSetup Interface

The **IAMovieSetup** interface provides methods that allow objects in a dynamic-link library (DLL) to be self-registering. The **IAMovieSetup** interface works in conjunction with an overall registration architecture that COM requires; this architecture is partially implemented in the DirectShow™ base classes. The remainder of the implementation is described in the following sections.

When to Implement

Implement this interface if you want your filter or plug-in distributor to be able to register itself automatically as part of a setup routine on an end user system. The two methods in this interface, IAMovieSetup::Register and IAMovieSetup::Unregister, are implemented by the CBaseFilter base class for self-registering filters. For a complete list of steps showing how to use this interface with the DirectShow class library, see Register DirectShow Objects.

When to Use

Use implemented methods on this interface from an entry point on the filter that is called by a setup utility or installation utility. These are used automatically by the DirectShow architecture and normally should not need to be called by any other component.

Methods in Vtable Order

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

IAMovieSetup methods	Description
Register	Adds the filter to the registry.
Unregister	Removes the filter from the registry.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMovieSetup::Register

IAMovieSetup Interface

Adds the filter to the registry.

HRESULT Register(void);

Return Values

Returns an HRESULT value that depends on the implementation. **HRESULT** can be one of the following standard constants, or other values not listed:

Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method isn't supported.
S_OK or NOERROR	Success.

Remarks

This method registers the filter, its pins, and the media type associated with the pins. It should be implemented to use IFilterMapper methods to accomplish this. See the CBaseFilter::Register member function for a description of its implementation.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMovieSetup::Unregister

IAMovieSetup Interface

Removes the filter from the registry.

HRESULT Unregister(void);

Return Values

Returns an HRESULT value that depends on the implementation. **HRESULT** can be one of the following standard constants, or other values not listed:

Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method isn't supported.
S_OK or NOERROR	Success.

Remarks

This method should be implemented to use the IFilterMapper::UnregisterFilter method to remove the filter from the registry. This effectively removes the pins and media types as well.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamConfig Interface

The **IAMStreamConfig** interface enables you to find out what types of formats an output pin can be connected with. Additionally it can be used to set stream formats, to tell a pin to connect with a certain format the next time it's connected, or to make it reconnect with a new format if it's already connected. Audio/video capture and audio/video compression filters implement this interface on their output pins, but potentially any filter dealing with audio or video can implement this interface on its output pins.

Use this interface to set a pin's output format, rather than connecting the pin by using a specific media type. After setting an output format, the pin will try to use that format the next time it connects. This enables you to call the IGraphBuilder::Render method on that pin and

get a desired format without connecting the pins and providing a CMediaType class object. Your pin should offer only the media type set in the CMediaType::SetFormat function in its enumeration of media types after SetFormat is called. Before then, offer media types as usual. This will ensure that the pin uses that format for connection. An application that needs to enumerate accepted media types using CBasePin::GetMediaType must do so before calling **SetFormat**.

The IAMStreamConfig::GetStreamCaps method can get more information about accepted media types than the traditional way of enumerating a pin's media types, so you typically should use it instead of pin enumeration. GetStreamCaps retrieves information about the kinds of audio and video formats allowed.

GetStreamCaps provides detailed information about the media types and capabilities supported by this pin. This method returns a set of structures that includes pairs of AM_MEDIA_TYPE and either a VIDEO_STREAM_CONFIG_CAPS or an AUDIO_STREAM_CONFIG_CAPS structures describing an accepted media type and how that media type can be altered to create other acceptable media types.

Note The cropping rectangle described throughout the **IAMStreamConfig** documentation is the same as the VIDEOINFOHEADER structure's rcSource rectangle for the output pin.

The output rectangle described throughout the **IAMStreamConfig** documentation is the same as the width and height members of the output pin's BITMAPINFOHEADER structure.

For more information on GetStreamCaps see Exposing Capture and Compression Formats.

When to Implement

Implement this interface on the video output pin when you are writing a video capture or video compression filter.

When to Use

Use this interface when your application or filter must get or set audio or video stream information.

WDM capture applications that wish to preview and then capture might have to set audio and video stream information on the preview pin and again on the capture pin.

Methods in Vtable Order

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

IAMStreamConfig methods	**Description**
SetFormat	Sets the audio or video stream's format.
GetFormat	Retrieves the audio or video stream's format.
GetNumberOfCapabilities	Retrieves the number of stream capabilities structures for the compressor.

GetStreamCaps	Obtains audio or video capabilities of a stream depending on which type of structure is pointed to in the *pSCC* parameter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamConfig::GetFormat

IAMStreamConfig Interface

Retrieves the audio or video stream's format.

HRESULT GetFormat(
AM_MEDIA_TYPE ***pmt*
);

Parameters

pmt
[out] Address of a pointer to an AM_MEDIA_TYPE structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Be sure to initialize the media type structure before using it. For example, the following code fragment calls the Win32® ZeroMemory function to initialize the structure.

```
AM_MEDIA_TYPE mt;
ZeroMemory(&mt, sizeof(mt))
GetFormat(&mt);
```

Call the FreeMediaType function to free the structure. cmt>

Previous Home Topic Contents Index Next

IAMStreamConfig::GetNumberOfCapabilities

IAMStreamConfig Interface

Retrieves the number of stream capabilities structures for the compressor.

HRESULT GetNumberOfCapabilities(
int **piCount*,
int **piSize*
);

Parameters

piCount
[out] Pointer to the number of VIDEO_STREAM_CONFIG_CAPS and/or AUDIO_STREAM_CONFIG_CAPS structures supported.
piSize
[out] Pointer to the size of the configuration structure (either AUDIO_STREAM_CONFIG_CAPS or VIDEO_STREAM_CONFIG_CAPS).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMStreamConfig::GetStreamCaps

IAMStreamConfig Interface

Obtains audio, video, or other capabilities of a stream depending on which type of structure is pointed to in the *pSCC* parameter.

HRESULT GetStreamCaps(
int *iIndex*,
AM_MEDIA_TYPE ***pmt*,
BYTE **pSCC*
);

Parameters

iIndex
[in] Index to the desired media type and capability pair. Use the GetNumberOfCapabilities method to retrieve the total number of these pairs. Possible index values range from zero to one less than the total number of pairs.

pmt
[out] Address of a pointer to an AM_MEDIA_TYPE structure.
pSCC
[out] Pointer to either a stream configuration structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method enables you to get more information about accepted media types rather than the traditional way of enumerating a pin's media types, so you typically should use it instead of pin enumeration. Information such as possible video capture rates, media types, and sizes is returned by the VIDEO_STREAM_CONFIG_CAPS structure. Audio capabilities of the filter's output pin, including the number of inputs, sampling rate, and bit rate granularity will be returned by an AUDIO_STREAM_CONFIG_CAPS structure.

Call DeleteMediaType to free the *pmt* media type.

For more information on **GetStreamCaps**, see Exposing Capture and Compression Formats.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamConfig::SetFormat

IAMStreamConfig Interface

Sets the audio or video stream's format.

```
HRESULT SetFormat(
  AM_MEDIA_TYPE *pmt
  );
```

Parameters

pmt
[in] Pointer to an AM_MEDIA_TYPE structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

A call to this method will fail if the pin is streaming.

If your output pin isn't connected and you can connect it with this media type, return S_OK from this method and start enumerating the specified media type as follows: Specify this format as format number zero in the CTransformOutputPin::GetMediaType function's *iPosition* parameter. You can offer and accept only this type to ensure that the pins will use this format for the connection when it occurs.

If your output pin is already connected and you can provide this type, then reconnect your pin. If the other pin can't accept the media type, fail this call and leave your connection alone.

IAMStreamControl Interface

The **IAMStreamControl** interface is exposed on input and output pins on any filter in a filter graph. This interface exposes methods that allow applications to control individual stream components in a filter graph. You can turn various streams on or off without affecting the rest of the graph. For example, you can turn off an audio stream while a video stream continues, for muting. Or a capture stream can be turned off while preview continues to flow. This interface also assists in frame accuracy when exact capture start or stop times are important.

Currently, the CBaseStreamControl base class implements **IAMStreamControl**. **CBaseStreamControl** enables the user to specify start and stop times in the CBaseStreamControl::StartAt and CBaseStreamControl::StopAt member functions and provides stream information in the CBaseStreamControl::GetInfo member function. **CBaseStreamControl** uses the StreamControlState enumerated data type to describe the various states a stream is in. A flowing stream is indicated by the STREAM_FLOWING setting; otherwise it is in a discarding state indicated by the STREAM_DISCARDING setting. See **StreamControlState** for more details on stream states.

If you want to implement this interface on your own your class should typically inherit from CBaseStreamControl to obtain an implementation of the CBaseStreamControl::StartAt, CBaseStreamControl::StopAt, and CBaseStreamControl::GetInfo methods. The **CBaseStreamControl** class also maintains state information and makes decisions about what to do with the sample. Developers implementing their own filters with pins that support **IAMStreamControl** through the **CBaseStreamControl** base class must follow certain guidelines outlined in the **CBaseStreamControl** documentation.

Note that there must be a clock in the filter graph or the stream control methods might not function as expected.

This interface is not available on the preview pin of capture cards with hardware overlay. Calling QueryInterface for this interface will return the error E_NOINTERFACE (0x80004002).

When to Implement

Implement on input or output pins of filters when you want precise control of the data stream. This interface enables you to turn off portions of the filter graph's streams at specific times without affecting the rest of the graph. Although this interface can be used throughout the graph, the output pins of audio and video capture filters and input pins of multiplexer filters primarily use it.

If you are writing a filter that will implement IAMStreamControl on one of its pins, you should set the STREAM_DISCARDING state so that the pin discards media samples in a timely fashion, rather than as soon as they are received. This means that if your pin is discarding samples as soon as it determines they are outside the time that the pin is supposed to be on, it will discard samples as fast as possible and the whole file could potentially be pushed into your filter and discarded in mere moments. This causes problems if the pin tries to call IAMStreamControl::StartAt at a later point in time because the entire file will have already been discarded. To avoid pins from dumping media samples as fast possible, your code should check the media sample's timestamp and wait until the reference clock verifies that the end of the sample's time has actually occurred before discarding. This is known as discarding in a timely fashion (see CBaseStreamControl for an implementation that does this).

When to Use

Use this interface to turn on or off certain portions of the filter graph's streams while other portions continue to process data. For example, your application can tell a video capture filter's output pin precisely when to start or stop capturing, independent of what is happening in the rest of the graph. This assists in frame accuracy when exact capture start or stop times are important.

Methods in Vtable Order

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

IAMStreamControl methods	Description
StartAt	Informs the pin when to start sending streaming data.
StopAt	Informs the pin when to suspend processing and supplying data.
GetInfo	Retrieves information about the current streaming settings.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamControl::GetInfo

IAMStreamControl Interface

Retrieves information about the current streaming settings.

HRESULT GetInfo(
AM_STREAM_INFO **pInfo*
);

Parameters

pInfo
[out] Pointer to an AM_STREAM_INFO structure that contains current stream settings.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Call this method to discover the state of the StreamControlState enumerated data type, which indicates the stream's state. Other values in the AM_STREAM_INFO structure include start time, stop time, start cookie, and stop cookie.

Previous Home Topic Contents Index Next

IAMStreamControl::StartAt

IAMStreamControl Interface

Informs the pin when to start sending streaming data.

HRESULT StartAt(
const REFERENCE_TIME * *ptStart*,
DWORD *dwCookie* **);**

Parameters

ptStart
[in] Time at which to start streaming as specified in the REFERENCE_TIME structure. If NULL, start immediately (no notification); if MAX_TIME, start canceled and will have no effect.
dwCookie
[in] Specifies a particular value to be sent with the notification when the start occurs. (Only used if *ptStart* is non-NULL or MAX_TIME).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Streams are enabled by default, so this method will have no effect unless a previous StopAt member function has been called.

If the pointer to the REFERENCE_TIME is not NULL or MAX_TIME, then pins should signal EC_STREAM_CONTROL_STARTED with an IPin pointer and the cookie specified in the *dwCookie* parameter. This enables applications to tie the events back to their requests. If the *ptStart* pointer is NULL or MAX_TIME, then the filter graph sends no event.

If start and stop are scheduled for a single point in time, the effect is as if the start occurred an infinitesimal time before the stop. You can use this to capture a single frame.

Previous Home Topic Contents Index Next

IAMStreamControl::StopAt

IAMStreamControl Interface

Informs the pin when to suspend processing and supplying data.

HRESULT StopAt(
const REFERENCE_TIME * *ptStop*,
BOOL *bSendExtra*,
DWORD *dwCookie* **);**

Parameters

ptStop
[in] Time at which to stop streaming as specified in the REFERENCE_TIME structure. If you specify NULL for *ptStop*, it will stop immediately (no notification); if MAX_TIME, cancels stop.
bSendExtra
[in] Indicates whether to send an extra sample after scheduled *ptStop* time.
dwCookie
[in] Specifies a particular value to send with the notification when the stop occurs (used only if *ptStart* if not NULL or MAX_TIME).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method is exposed by pins that support the stopping of streams. It sets the StreamControlState enumeration type to STREAM_DISCARDING.

In video capture, you would typically call **StopAt** on both the output pin of a capture filter and the input pin of a multiplexer, and pay attention only to the notification from the multiplexer. This ensures that the capture filter doesn't needlessly capture extra frames, while guaranteeing that the multiplexer has, in fact, saved the last frame to a file.

In addition, you should specify TRUE for the *bSendExtra* parameter on the capture pin, and specify FALSE to the multiplexer pin. If an extra frame is not sent, the multiplexer will wait for the stop time indefinitely and not realize it already has received all the capture information. The multiplexer will discard the extra sample sent by the capture pin, so it will not get written to the file. Do not set *bSendExtra* to TRUE unless you also use IAMStreamControl on another downstream pin too, like in the preceding case.

If you call **StopAt** with a time that is in the middle of a packet, the filter will deliver the whole packet before going into a discarding state. Also, if start and stop are scheduled for a single point in time, the effect is as if the start occurred an infinitesimal time before the stop. You can use this effect to capture a single frame (see CBaseStreamControl for an implementation example).

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamSelect Interface

The **IAMStreamSelect** interface controls which logical streams are played and retrieves information about them.

When to Implement

Implement this interface on your filter when you want to enable selection of logical streams and provide information about them. An example of logical stream selection is selection from a set of audio streams that encode different national languages. Perhaps you could choose English from among a set of audio streams that include English, German, and French. The MPEG splitter implements this interface.

When to Use

Use this interface when you want to select between available streams; for example, when you want to select the streams for a particular locale.

Methods in Vtable Order

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

IAMStreamSelect methods	Description
Count	Retrieves the total count of available streams.
Info	Retrieves information about a given stream.
Enable	Enables or disables a given stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamSelect::Count

IAMStreamSelect Interface

Retrieves the total count of available streams.

```
HRESULT Count(
  DWORD *pcStreams
  );
```

Parameters

pcStreams
[out] Pointer to a value indicating the number of available streams.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns S_OK.

Previous Home Topic Contents Index Next

IAMStreamSelect::Enable

IAMStreamSelect Interface

Enables or disables a given stream.

HRESULT Enable(
long *lIndex*,
DWORD *dwFlags*
);

Parameters

lIndex
[in] Index number of desired stream. Zero-based.
dwFlags
[in] Flag indicating whether to enable or disable the stream. Valid values include the following:

Value	Meaning
Zero	Disable all streams in the group containing this stream.
AMSTREAMSELECTENABLE_ENABLE	Enable only this stream within the given group and disable all others.
AMSTREAMSELECTENABLE_ENABLEALL	Enable all streams in the group containing this stream.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_NOTIMPL if support for the specified flag has not been implemented, E_INVALIDARG if the stream ID is invalid, or S_OK otherwise.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMStreamSelect::Info

IAMStreamSelect Interface

Retrieves information about a given stream.

HRESULT Info(
long *lIndex*,
AM_MEDIA_TYPE ***ppmt*,

DWORD **pdwFlags*,
LCID **plcid*,
DWORD **pdwGroup*,
WCHAR ***ppszName*,
IUnknown ***ppObject*,
IUnknown ***ppUnk*
);

Parameters

lIndex
[in] Index number of desired stream. Zero-based.

ppmt
[out] Address of a pointer to the stream's media type. Optional. Use the DeleteMediaType function to free the AM_MEDIA_TYPE structure when done.

pdwFlags
[out] Pointer to flags. Optional. Valid values include the following:

Value	Meaning
Zero	Disable this stream.
AMSTREAMSELECTINFO_ENABLED	Enable the stream.
AMSTREAMSELECTINFO_EXCLUSIVE	Turns off the other streams in the group when enabling this one.

plcid
[out] Pointer to the locale context (LCID) value. This parameter points to a zero value if there is no LCID. Optional.

pdwGroup
[out] Pointer to the logical group. Optional.

ppszName
[out] Pointer to the stream name. Optional. Free with the CoTaskMemFree function when done.

ppObject
[out] Pointer to the pin or filter object associated with this stream. Optional. The object can change if the IAMStreamSelect::Enable method is called. This parameter contains a null value upon return from this method if there is no associated object.

ppUnk
[out] Address of a pointer to a stream-specific interface.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns S_FALSE if *lIndex* is out of range, or S_OK otherwise.

Remarks

The first stream in each group is the default.

IAMTimecodeDisplay Interface

The **IAMTimecodeDisplay** interface contains properties and methods that define behavior of an external SMPTE/MIDI timecode display device. This interface should be implemented in combination with IAMExtDevice, IAMExtTransport, and the other timecode interfaces to control an external device, such as a VCR, which can read, generate and/or display timecode data. This interface controls the physical timecode character generator display that is either built into a VCR or is on some other similar external device.

For more information on SMPTE timecode see the Control an External Device in DirectShow overview article.

When to Implement

Implement this interface on an external device filter that will control the timecode display on an external timecode reader or generator. Timecode readers or generators can be built into a VCR or can be a separate external device. Do not try to implement this interface if your external device can't display timecode or if your timecode is being generated through an internal card with no integral or overlay hardware.

This interface is not intended for rendering in a DirectShow filter graph, it is purely for use on external device displays.

When to Use

Use this interface when applications need to control an external device and how it's timecode information is displayed.

Hardware Requirements

See the IAMExtTransport interface for hardware requirements.

Methods in Vtable Order

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

IAMTimecodeDisplay methods	Description
GetTCDisplayEnable	Determines whether an external device's timecode character generator output is enabled or disabled.
SetTCDisplayEnable	Enables or disables an external device's timecode character output generator.
GetTCDisplay	Retrieves current settings of the timecode character generator output.
SetTCDisplay	Sets the timecode character generator output characteristics.

Previous Home Topic Contents Index Next

IAMTimecodeDisplay::GetTCDisplay

IAMTimecodeDisplay Interface

Retrieves current settings of the timecode character generator output.

```
HRESULT GetTCDisplay(
  long Param,
  long *pValue
  );
```

Parameters

Param

[in] Timecode display characteristic. Specify one of the following items you want to get settings for.

Value	Meaning
ED_TCD_BORDER	White border for black characters, black border for white characters
ED_TCD_INTENSITY	Intensity (brightness) of characters
ED_TCD_INVERT	Black characters on white background or white characters on black background
ED_TCD_POSITION	Position of characters
ED_TCD_SIZE	Size of characters
ED_TCD_SOURCE	Source of display's data
ED_TCD_TRANSPARENCY	Transparency of characters

pValue

[out] Current setting of the parameter specified in *Param*. This parameter retrieves one of the following values:

Value	Meaning
If ED_TCD_SOURCE specified in *Param*, will return one of the following:	
ED_TCG	TimeCode generator
ED_TCR	TimeCode reader
If , ED_TCD_SIZE specified in *Param*, will return one of the following:	
ED_LARGE	Large
ED_MED	Medium

ED_SMALL	Small
If ED_TCD_POSITION specified in *Param*, will return one of the following:	
ED_BOTTOM	Bottom
ED_MIDDLE	Middle
ED_TOP	Top
in combination with:	
ED_CENTER	Center
ED_LEFT	Left
ED_RIGHT	Right
If ED_TCD_INTENSITY specified in *Param*, will return one of the following:	
ED_HIGH	High
ED_LOW	Low
If ED_TCD_TRANSPARENCY is specified in *Param*, will return a value from 0 to 4, 0 being completely opaque.	
If ED_TCD_INVERT specified in *Param*, will return one of the following:	
OAFALSE	Black characters on white background
OATRUE	White characters on black background
If ED_TCD_BORDER specified in *Param*, will return one of the following:	
OAFALSE	Black characters for white border
OATRUE	White border for black characters

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

See Also

IAMTimecodeDisplay::SetTCDisplay

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeDisplay::GetTCDisplayEnable

IAMTimecodeDisplay Interface

Determines whether an external device's timecode character generator output is enabled or disabled.

HRESULT GetTCDisplayEnable(
long **pState*
);

Parameters

pState
[out] OATRUE specifies enabled; OAFALSE specifies disabled.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method is not intended for character rendering inside a filter graph, it is purely intended for hardware displays. Ensure that your external timecode reader or generator has display capability before trying to use this method.

See Also

IAMTimecodeDisplay::SetTCDisplayEnable

Previous Home Topic Contents Index Next

IAMTimecodeDisplay::SetTCDisplay

IAMTimecodeDisplay Interface

Sets the timecode character generator output characteristics.

HRESULT SetTCDisplay(
long *Param*,
long *Value*
);

Parameters

Param

[in] Timecode display characteristic. Specify one of the following properties you want to set properties for.

Value	Meaning
ED_TCD_BORDER	White border for black characters, black border for white characters
ED_TCD_INTENSITY	Intensity (brightness) of characters
ED_TCD_INVERT	Black characters on white background or white characters on black background
ED_TCD_POSITION	Position of characters
ED_TCD_SIZE	Size of characters
ED_TCD_SOURCE	Source of the display's data
ED_TCD_TRANSPARENCY	Transparency of characters

Value

[in] Setting of the parameter specified in *Param*. Must be one of the following:

Value	Meaning
If ED_TCD_SOURCE specified in *Param*, set one of the following:	
ED_TCG	TimeCode generator
ED_TCR	TimeCode reader
If , ED_TCD_SIZE specified in *Param*, set one of the following:	
ED_LARGE	Large
ED_MED	Medium
ED_SMALL	Small
If ED_TCD_POSITION specified in *Param*, set one of the following:	
ED_BOTTOM	Bottom
ED_MIDDLE	Middle
ED_TOP	Top
In combination with:	
ED_CENTER	Center
ED_LEFT	Left
ED_RIGHT	Right
If ED_TCD_INTENSITY specified in *Param*, set one of the following:	
ED_HIGH	High
ED_LOW	Low
If ED_TCD_TRANSPARENCY specified in *Param*, set a value from 0 to 4, 0 being completely opaque, 4 being as dark as possible.	
If ED_TCD_INVERT specified in *Param*, set one of the following:	
OAFALSE	Black on white
OATRUE	White on black
If ED_TCD_BORDER specified in *Param*, set one of the following:	
OAFALSE	Black characters for white border
OATRUE	White border for black characters

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

See Also

IAMTimecodeDisplay::GetTCDisplay

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeDisplay::SetTCDisplayEnable

IAMTimecodeDisplay Interface

Enables or disables an external device's timecode character output generator.

```
HRESULT SetTCDisplayEnable(
  long State
  );
```

Parameters

State
[in] Specify OATRUE to enable; OAFALSE to disable.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method is not intended for rendering characters inside a filter graph, it is purely intended for hardware displays. Ensure that your external timecode reader or generator has display capability before trying to use this method.

See Also

IAMTimecodeDisplay::GetTCDisplayEnable

Previous Home Topic Contents Index Next

IAMTimecodeGenerator Interface

The **IAMTimecodeGenerator** interface contains properties and methods that specify how an external SMPTE/MIDI timecode generator should supply data to the filter graph and the formats in which timecode should be supplied. This interface should be implemented in combination with the IAMExtDevice and IAMExtTransport interfaces to control an external device, such as a VCR. This interface provides methods that enable applications to specify various SMPTE/MIDI timecode modes or formats that an external device should use in the generation of timecode, and methods that verify that the generator is working properly.

SMPTE timecode is a frame addressing system that identifies video and audio sources, makes automatic track synchronization possible, and provides a container for additional data related to the production. SMPTE timecode's main purpose is to provide a machine-readable address for video and audio. It is displayed in hh:mm:ss:ff format and is thoroughly defined in ANSI/SMPTE 12-1986.

For more information on SMPTE timecode see the Control an External Device in DirectShow overview article.

See the IAMTimecodeReader interface for more information on methods which access an external timecode reader.

When to Implement

Implement this interface on an external device filter when you want to control how SMPTE/MIDI timecode information is generated by an external timecode generator.

Expose the IMediaSeeking interface on your filter to enable applications to convert timecode to DirectShow reference time (by using IMediaSeeking::ConvertTimeFormat).

Your external device must be able to read timecode and send it to the computer over its control interface (see hardware requirements). If this is not the case, you must either have a timecode reader card in your computer, or you can write a software decoder that converts VITC embedded in captured video frames or LTC captured as an audio signal into DirectShow timecode samples.

When to Use

Use this interface when you want to generate SMPTE timecode in an external device.

Hardware Requirements

See the IAMExtTransport interface for hardware requirements.

Methods in Vtable Order

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

IAMTimecodeGenerator methods	Description
GetTCGMode	Retrieves the SMPTE timecode generator properties.
SetTCGMode	Sets the SMPTE timecode generator properties.
put_VITCLine	Specifies which line(s) to insert the vertical interval timecode information into.
get_VITCLine	Retrieves which line(s) the vertical interval timecode information has been inserted into.
SetTimecode	Sets the timecode, userbit value, or both.
GetTimecode	Retrieves the most recent timecode and/or userbit value available in the stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeGenerator::GetTCGMode

IAMTimecodeGenerator Interface

Retrieves the SMPTE timecode generator properties.

```
HRESULT GetTCGMode(
  long Param,
  long *pValue
  );
```

Parameters

Param

[in] Timecode generator mode. Specify one of the following modes you want to get settings for.

Value	Meaning
ED_TCG_FRAMERATE	Frame rate
ED_TCG_REFERENCE_SOURCE	Source of the count value
ED_TCG_SYNC_SOURCE	Source of the hardware clock reference
ED_TCG_TIMECODE_TYPE	SMPTE timecode format of the generator

pValue

[out] Current setting of the parameter specified in *Param*.

If you specify ED_TCG_TIMECODE_TYPE in *Param*, this parameter retrieves one of the following:

Value	Meaning
ED_TCG_MIDI_FULL	MIDI full frame timecode
ED_TCG_MIDI_QF	MIDI quarter frame timecode
ED_TCG_SMPTE_LTC	Linear timecode
ED_TCG_SMPTE_VITC	Vertical interval timeCode

If you specify ED_TCG_FRAMERATE in *Param*, this parameter retrieves one of the following:

Value	Meaning
ED_FORMAT_SMPTE_24	24 frames per second
ED_FORMAT_SMPTE_25	25 frames per second
ED_FORMAT_SMPTE_30	30 frames per second. Nondrop frame
ED_FORMAT_SMPTE_30DROP	30 frames per second. Drop frame (actually 29.97 fps)

If you specify ED_TCG_SYNC_SOURCE in *Param*, this parameter retrieves one of the following:

Value	Meaning
ED_TCG_FREE	Lock to nothing (freerun)
ED_TCG_READER	Lock to timecode reader
ED_TCG_VIDEO	Lock to incoming video

If you specify ED_TCG_REFERENCE_SOURCE in *Param*, this parameter retrieves one of the following:

Value	Meaning
ED_TCG_FREE	No count reference source
ED_TCG_READER	Syncronize to reader value (jamsync)

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method returns various settings of the timecode generator. For more information on ED_TCG_TIMECODE_TYPE, see IAMTimecodeReader::SetTCRMode.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeGenerator::GetTimecode

IAMTimecodeGenerator Interface

Retrieves the most recent timecode and/or userbit value available in the stream.

HRESULT GetTimecode(
PTIMECODE_SAMPLE *pTimecodeSample*
);

Parameters

pTimecodeSample
[out] Pointer to a TIMECODE_SAMPLE timecode structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Use this method to obtain the most recent timecode value available in the stream. The application can use this to monitor the timecode and verify the generator is working properly.

See Also

IAMTimecodeGenerator::SetTimecode

Previous Home Topic Contents Index Next

IAMTimecodeGenerator::get_VITCLine

IAMTimecodeGenerator Interface

Retrieves which line(s) the vertical interval timecode information has been inserted into.

HRESULT get_VITCLine(
long **pLine* **);**

Parameters

pLine
[out] Pointer to the vertical line(s) containing the timecode information (valid lines are 11-20).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

To get VITC information from multiple lines, make successive calls to this method, once for each line desired, with the hi bit set for each line.

See Also

IAMTimecodeGenerator::put_VITCLine

IAMTimecodeGenerator::put_VITCLine

IAMTimecodeGenerator Interface

Specifies which line to insert the vertical interval timecode information into.

HRESULT put_ VITCLine(
long *Line* **);**

Parameters

Line
[in] Vertical line to contain the timecode information (valid lines are 11-20; 0 means autoselect).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

To generate VITC on specific multiple lines, make successive calls to this method, once for each line desired.

Set the hi bit to add to this line to any previously set lines.

See Also

IAMTimecodeGenerator::get_VITCLine

IAMTimecodeGenerator::SetTCGMode

IAMTimecodeGenerator Interface

Sets the SMPTE timecode generator properties.

HRESULT SetTCGMode(
long *Param*,
long *Value*
);

Parameters

Param

[in] Timecode generator mode. Specify one of the following properties.

Value	Meaning
ED_TCG_FRAMERATE	Frame rate
ED_TCG_REFERENCE_SOURCE	Source of the count value
ED_TCG_SYNC_SOURCE	Source of the hardware clock reference
ED_TCG_TIMECODE_TYPE	SMPTE timecode format of the generator

Value

[in] Setting of the parameter specified in *Param*.

Value	Meaning
If ED_TCG_TIMECODE_TYPE specified in *Param*, set one of the following:	
ED_TCG_MIDI_FULL	MIDI Full Frame timecode
ED_TCG_MIDI_QF	MIDI Quarter Frame timecode
ED_TCG_SMPTE_LTC	Linear TimeCode
ED_TCG_SMPTE_VITC	Vertical Interval TimeCode
If , ED_TCG_FRAMERATE specified in *Param*, this parameter is set to one of the following:	
ED_FORMAT_SMPTE_24	24 frames per second
ED_FORMAT_SMPTE_25	25 frames per second
ED_FORMAT_SMPTE_30	30 frames per second. Nondrop frame
ED_FORMAT_SMPTE_30DROP	30 frames per second. Drop frame (actually 29.97 fps)
If ED_TCG_SYNC_SOURCE specified in *Param*, set one of the following:	
ED_TCG_FREE	Lock to nothing (freerun)
ED_TCG_READER	Lock to timecode reader
ED_TCG_VIDEO	Lock to incoming video
If ED_TCG_REFERENCE_SOURCE specified in *Param*, set one of the following:	

ED_TCG_FREE	No count reference source
ED_TCG_READER	sync to reader value (jamsync)

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method sets various properties of the timecode generator. For more information on ED_TCG_TIMECODE_TYPE, see the IAMTimecodeReader::SetTCRMode method.

See Also

IAMTimecodeGenerator::GetTCGMode

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeGenerator::SetTimecode

IAMTimecodeGenerator Interface

Sets the timecode, userbits value, or both.

```
HRESULT SetTimecode(
  PTIMECODE_SAMPLE pTimecodeSample
  );
```

Parameters

pTimecodeSample
[in] Pointer to a TIMECODE_SAMPLE timecode structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

To set only timecode, set userbit value to NULL, and vice versa. If generator is running, these values will take effect immediately.

See Also

IAMTimecodeGenerator::GetTimecode

IAMTimecodeReader Interface

IAMTimecodeReader is an interface that can be implemented to read SMPTE (Society of Motion Picture and Television Engineers) or MIDI timecode from an external device. It contains properties and methods that specify the timecode format that an external device should read and how it is embedded in the media. It is expected that you will use this interface with the IAMExtDevice and IAMExtTransport interfaces to control an external device, such as a VCR, which can read timecode data.

SMPTE timecode is a frame addressing system that identifies video and audio sources, makes automatic track synchronization possible, and provides a container for additional data related to the source material. SMPTE timecode's main purpose is to provide a machine-readable address for video and audio. It is displayed in hh:mm:ss:ff (hours, minutes, seconds, frames) format and is thoroughly defined in ANSI/SMPTE 12-1986.

For more information on SMPTE timecode see the Control an External Device in DirectShow overview article.

When to Implement

Implement this interface on an external device filter when you want to specify how an external device should read SMPTE/MIDI timecode information.

Expose the IMediaSeeking interface on your filter so that applications can convert timecode to DirectShow reference time (by using the IMediaSeeking::ConvertTimeFormat method).

Your external device must be able to read timecode and send it to the computer over its control interface. If this is not the case, you must either have a timecode reader card in your computer, or you can write a software decoder that converts VITC (Vertical Interval Timecode) in captured video frames or LTC (Linear Timecode) captured as an audio signal into DirectShow timecode samples.

When to Use

Use this interface when you need to read timecode information for controlling an external device, or when you want to use timecode information from an external device in applications that must refer to original program information.

Applications generally save timecode in one of two ways. It is either written to the capture file as an additional stream or as a discontinuity table stored in the extended AVI file index. It is

commonly used to trigger capture or playback and to create edit decision lists that describes how source material is organized into a finished product.

If you intend to capture timecode, treat it as a separate stream that has its own media type. It can be consumed by an appropriate file-writing multiplexer filter. However, sometimes there are errors in reading the timecode off the tape because of dropouts and other mechanical tape problems. In such cases, the timecode source filter should simply drop samples and mark the next valid one with the discontinuity property.

If you intend to use timecodes to trigger capture or playback from a timecoded (or "striped") videotape, the sequence of events goes as follows:

1. Build a capture graph, open a target AVI file, and preallocate disk space if necessary. If the captured material will be appended to an existing AVI file, seek to the end of the file before writing. The capture graph is paused at this point.
2. Search the VCR to the capture start point and note the timecode. You can either enter this value manually into your program, or the application can automatically read it. Automatic reading requires that the graph is running but the stream control interfaces on the file multiplexer's input pins are discarding incoming samples, effectively gating the capture.
3. Cue the VCR to preroll position, usually five seconds before the target point.
4. Start the VCR and the graph. When the trigger point is reached (or the trigger point minus the file writer's preroll), the stream control interfaces release the file multiplexer and it begins streaming media samples to the file writer.
5. You can stop the capture process manually or by setting a duration property on the stream control interface.

You must consider discontinuous timecode, both during preroll and during the capture process; it is reasonable to demand that the timecode be continuous and monotonically increasing throughout the preroll and capture start point. This prevents a potentially ambiguous calculation of relative stream times by the IMediaSeeking::ConvertTimeFormat method. Also, the timecode need not be the only gating signal for triggered capture. Any time-stamped data stored in the vertical blanking interval, such as Intercast or Closed Caption data (XDS), can be used to start the streaming of video and audio data to disk.

Hardware Requirements

See the IAMExtTransport interface for hardware requirements.

Methods in Vtable Order

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

IAMTimecodeReader methods	Description
GetTCRMode	Retrieves properties of the timecode reader.
SetTCRMode	Sets the timecode reader properties.
put_VITCLine	Specifies the vertical interval line that the timecode reader will use to read timecode.
get_VITCLine	Retrieves the vertical interval line that the timecode reader is using to read timecode.

GetTimecode	Retrieves the most recent timecode, userbits, and flag values available in the stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTimecodeReader::GetTCRMode

IAMTimecodeReader Interface

Retrieves the timecode reader's properties.

```
HRESULT GetTCRMode(
  long Param,
  long *pValue
  );
```

Parameters

Param
[in] Timecode reader property to get (use ED_TCR_SOURCE).

pValue
[out] Value of the requested timecode reader property. Must be one of the following:

Value	Meaning
ED_TCR_CT	Control track
ED_TCR_LTC	Linear timecode
ED_TCR_VITC	Vertical interval timecode

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Linear TimeCode is recorded on an analog audio track as a bi-phase mark-encoded signal. Each timecode frame is one video frame time in duration.

Vertical TimeCode is usually stored in two lines of a video signal's vertical interval, somewhere between lines 11 and 20.

Control Track is a once-per-frame signal recorded on a special track on a tape. The head and drive servo mechanisms use it to keep everything locked. It is also used to drive the counter on machines without timecode capability, and can optionally be used on machines equipped with a timecode reader.

See Also

IAMTimecodeReader::SetTCRMode

Previous Home Topic Contents Index Next

IAMTimecodeReader::GetTimecode

IAMTimecodeReader Interface

Retrieves the most recent timecode, userbit, and flag values available in the stream.

HRESULT GetTimecode(
PTIMECODE_SAMPLE *pTimecodeSample*
);

Parameters

pTimecodeSample
[out] Pointer to a TIMECODE_SAMPLE timecode structure.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Use this method to monitor the timecode and to parse duplicates and discontinuities. The source filter supplying the timecode, or possibly a downstream filter, might want to parse for discontinuities or errors since you have to look at every sample to be able to retrieve the most recent timecode.

Applications can fill undefined bits in the timecode word to store syncronization information, or to encode original film and audio tape information. These undefined bits, or userbits, are retrieved by calling this method.

IAMTimecodeReader::get_VITCLine

IAMTimecodeReader Interface

Retrieves the vertical interval line that the timecode reader is using to read timecode.

HRESULT get_VITCLine(
long *pLine* **);**

Parameters

pLine
[out] Vertical line containing timecode information (valid lines are 11-20).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

The hi bit indicates that multiple lines are used and successive calls will cycle through the line numbers.

See Also

IAMTimecodeReader::put_VITCLine

Previous Home Topic Contents Index Next

IAMTimecodeReader::put_VITCLine

IAMTimecodeReader Interface

Specifies the vertical interval line that the timecode reader will use to read timecode.

HRESULT put_VITCLine(
long *Line* **);**

Parameters

Line
[in] Vertical line containing timecode information (valid lines are 11-20; 0 means

autoselect).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

If VITC mode is specified in the IAMTimecodeReader::SetTCRMode method, you must specify which line or lines will contain timecode information. To read VITC on specific multiple lines, the caller would make successive calls to **put_VITCLine**, once for each line desired.

Set the hi bit to add to the list of lines for readers that test across multiple lines.

See Also

IAMTimecodeReader::get_VITCLine

IAMTimecodeReader::SetTCRMode

IAMTimecodeReader Interface

Sets the timecode reader properties.

HRESULT SetTCRMode(
long *Param*,
long *Value*
);

Parameters

Param
[in] Property you want to set (use ED_TCR_SOURCE).
Value
[in] Value of the specified property; currently one of the following:

Value	Meaning
ED_TCR_CT	Control Track
ED_TCR_LTC	Linear TimeCode
ED_TCR_VITC	Vertical Interval TimeCode

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Linear TimeCode is recorded on an analog audio track as an NRZ bi-phase mark-encoded signal. Each timecode frame is one video frame time in duration.

Vertical TimeCode is usually stored in two lines of a video signal's vertical interval, somewhere between 10 and 20.

Control Track is a once-per-frame signal recorded on a special track on a tape. The head and drive servo mechanisms use it to keep everything locked. It is also used to drive the counter on machines without timecode capability, and can optionally be used on machines equipped with a timecode reader.

See Also

IAMTimecodeReader::GetTCRMode

Previous Home Topic Contents Index Next

IAMTVTuner Interface

The **IAMTVTuner** interface is implemented on filters that provide TV tuning capabilities. A TV tuner filter is a device that selects an analog broadcast or cable channel to be viewed. The **IAMTVTuner** interface enables applications to set these transmission types through the TunerInputType enumerated data type.

Because Microsoft® Video for Windows® wasn't written with TV tuning capabilities in mind, you can implement TV tuner filters only on operating systems that can interpret TV tuning information. The Windows Driver Model implements a version that contains international channel to frequency mapping tables, found in the Country Codes and Channel to Frequency Mappings appendix, which you can use in a filter graph.

The **IAMTVTuner** interface supports multistandard analog decoders, which you can enumerate and select by using the get_AvailableTVFormats method. The AnalogVideoStandard data type contains these formats, which include NTSC, PAL, and SECAM, among others. **IAMTVTuner** also supports tuners with multiple input pins, to allow for multiple devices and multiple transmission types.

IAMTVTuner also maps TV channels to specific frequencies through the IAMTVTuner::put_Channel and IAMTVTuner::AutoTune methods. These methods handle the details of the conversion so that the hardware driver receives an exact frequency. Because channels in different countries map to different frequencies, worldwide mapping tables are provided in the Country Codes and Channel to Frequency Mappings appendix. Override the

existing country code by selecting the new value from the appendix and passing it in as the parameter for the IAMTVTuner::put_CountryCode method. This is useful when a country wants to receive broadcast video from a different national source.

When to Implement

Implement this interface when you write a filter that can tune a TV.

When to Use

Use this interface when setting TV channels and to get or set information about their frequencies. This interface can also determine what analog video standards your TV supports.

Methods in Vtable Order

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

IAMTVTuner methods	Description
get_AvailableTVFormats	Retrieves all the analog video TV standards that are supported by the tuner.
get_TVFormat	Retrieves the current analog video TV standard in use.
put_Channel	Sets the TV channel.
get_Channel	Retrieves the current TV channel set by put_Channel.
ChannelMinMax	Retrieves the highest and lowest channels available.
AutoTune	Scans for a precise signal on the channel's frequency.
StoreAutoTune	Saves the fine-tuning information for all channels.
put_CountryCode	Sets the country code to establish the frequency to use.
get_CountryCode	Retrieves the country code that establishes the current channel to frequency mapping.
put_TuningSpace	Sets a storage index for regional channel to frequency mappings.
get_TuningSpace	Retrieves the storage index for regional fine tuning set in put_TuningSpace.
get_NumInputConnections	Retrieves the number of TV sources plugged into the tuner filter.
put_InputType	Sets the tuner input type (cable or antenna).
get_InputType	Retrieves the input type (Cable or Antenna) set in put_InputType.
put_ConnectInput	Sets the hardware tuner input connection.
get_ConnectInput	Retrieves the hardware tuner input connection.
get_VideoFrequency	Retrieves the current video frequency.
get_AudioFrequency	Retrieves the current audio frequency.

IAMTVTuner::AutoTune

IAMTVTuner Interface

Scans for a precise signal on the channel's frequency.

HRESULT AutoTune(
long *lChannel*,
long * *plFoundSignal*
);

Parameters

lChannel
[in] TV channel number.
plFoundSignal
[out] Value indicating whether the channel's frequency was found; TRUE indicates found, FALSE indicates not found.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

TV channels generally map to a unique frequency depending on regional variances. To avoid interference between multiple transmitters that are assigned the same channel when they are in close geographic proximity, small frequency offsets are introduced at each transmitter. In the US, this offset ranges up to +/– 26.25 kilohertz (kHz).

This method handles the channel to frequency conversion and scans for the most precise frequency. Store these values by calling the IAMTVTuner::StoreAutoTune method. Base frequencies for channels can be found in the Country Codes and Channel to Frequency Mappings appendix.

Previous Home Topic Contents Index Next

IAMTVTuner::ChannelMinMax

IAMTVTuner Interface

Retrieves the highest and lowest channels available.

HRESULT ChannelMinMax(
long **lChannelMin*,
long **lChannelMax*
);

Parameters

lChannelMin
[out] Pointer to the lowest channel.
lChannelMax
[out] Pointer to the highest channel.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Frequencies for channels are found in the Country Codes and Channel to Frequency Mappings appendix.

Previous Home Topic Contents Index Next

IAMTVTuner::get_AudioFrequency

IAMTVTuner Interface

Retrieves the currently tuned audio frequency.

HRESULT get_ AudioFrequency(
long **lFreq*
);

Parameters

lFreq
[out] Pointer to the audio frequency.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::get_AvailableTVFormats

IAMTVTuner Interface

Retrieves all the analog video TV standards that are supported by the tuner.

HRESULT get_ AvailableTVFormats(
long **lAnalogVideoStandard*
);

Parameters

lAnalogVideoStandard
[out] Pointer to the combination of analog video standards supported.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

See the AnalogVideoStandard enumerated data type for supported formats.

Previous Home Topic Contents Index Next

IAMTVTuner::get_Channel

IAMTVTuner Interface

Retrieves the current TV channel set by put_Channel.

HRESULT get_Channel (
long * *plChannel*,
long **plVideoSubChannel*,
long **plAudioSubChannel*
);

Parameters

plChannel
[out] Pointer to the channel.

plVideoSubChannel
[out] Pointer to a predefined video subchannel value. Specify AMTUNER_SUBCHAN_NO_TUNE for no tuning or AMTUNER_SUBCHAN_DEFAULT for default subchannel.
plAudioSubChannel
[out] Pointer to a predefined audio subchannel value. Specify AMTUNER_SUBCHAN_NO_TUNE for no tuning or AMTUNER_SUBCHAN_DEFAULT for default subchannel.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

See the Country Codes and Channel to Frequency Mappings appendix for frequencies for *plChannel*.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTVTuner::get_ConnectInput

IAMTVTuner Interface

Retrieves the hardware tuner input connection.

HRESULT get_ConnectInput (
long **plIndex*
);

Parameters

plIndex
[out] Pointer to the input pin to get the connection for.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::get_CountryCode

IAMTVTuner Interface

Retrieves the country code that establishes the current channel to frequency mapping.

HRESULT get_CountryCode (
long * *plCountryCode*
);

Parameters

plCountryCode
[in] Country code currently in use by the TV Tuner filter.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

The IAMTVTuner::put_CountryCode method determines which channel to frequency mapping table to use. This establishes the base frequencies for the given country. Use the IAMTVTuner::AutoTune method to determine the exact frequencies for specific regions.

Override the country code when a country wants to receive broadcast video from a different national source. See the Country Codes and Channel to Frequency Mappings appendix for a list of country codes.

Previous Home Topic Contents Index Next

IAMTVTuner::get_InputType

IAMTVTuner Interface

Retrieves the input type set in put_InputType.

HRESULT get_InputType (
long *lIndex*,
TunerInputType * *pInputType*
);

Parameters

lIndex
[in] Index value that specifies the input pin that will be set.
pInputType
[out] Pointer to the TunerInputType connection type; either cable (TunerInputCable) or antenna (TunerInputAntenna).

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::get_NumInputConnections

IAMTVTuner Interface

Retrieves the number of TV sources plugged into the tuner filter.

HRESULT get_ NumInputConnections(
long * *plNumInputConnections*
);

Parameters

plNumInputConnections
[out] Number of TV sources plugged into the tuner filter.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::get_TuningSpace

IAMTVTuner Interface

Gets the storage index for regional fine tuning set in put_TuningSpace.

HRESULT get_TuningSpace(
long * *plTuningSpace*
);

Parameters

plTuningSpace
[out] Value specifying the current locale.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

As TV tuners move into portable systems, you must retain locale-specific mappings of available channels and their actual frequencies. Formulating different *lTuningSpace* values for each locale provides a way of switching the channel/frequency mappings when moving from region to region.

Previous Home Topic Contents Index Next

IAMTVTuner::get_TVFormat

IAMTVTuner Interface

Retrieves the current analog video TV standard in use.

HRESULT get_ TVFormat(
long * *plAnalogVideoStandard* **);**

Parameters

plAnalogVideoStandard
[out] Pointer to the analog video standard currently in use by the TV Tuner filter.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

See the AnalogVideoStandard enumerated data type for supported formats.

IAMTVTuner::get_VideoFrequency

IAMTVTuner Interface

Retrieves the current video frequency.

HRESULT get_VideoFrequency(
long **lFreq*
);

Parameters

lFreq
[out] Pointer to the video frequency.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

IAMTVTuner::put_Channel

IAMTVTuner Interface

Sets the TV channel.

HRESULT put_ Channel(
long *lChannel*,
long *lVideoSubChannel*,
long *lAudioSubChannel*
);

Parameters

lChannel

[in] TV channel number.

lVideoSubChannel

Predefined video subchannel value. Specify AMTUNER_SUBCHAN_NO_TUNE for no tuning or AMTUNER_SUBCHAN_DEFAULT for default subchannel.

lAudioSubChannel

Predefined audio subchannel value. Specify AMTUNER_SUBCHAN_NO_TUNE for no tuning or AMTUNER_SUBCHAN_DEFAULT for default subchannel.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method handles the channel to frequency function call that converts the TV channel to a TV frequency. Frequencies for channels are found in the Country Codes and Channel to Frequency Mappings appendix.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMTVTuner::put_ConnectInput

IAMTVTuner Interface

Sets the hardware tuner input connection.

```
HRESULT put_ ConnectInput(
  long lIndex
  );
```

Parameters

lIndex

[in] Index value of the input pin to set connection for.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::put_CountryCode

IAMTVTuner Interface

Sets the country code to establish the frequency to use.

HRESULT put_ CountryCode(
long *lCountryCode*
);

Parameters

lCountryCode
[in] Value indicating the country code.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method establishes the base frequencies for the given country. Use the IAMTVTuner::AutoTune method to determine the exact frequencies for specific regions, unless there are previously cached settings for the new country.

Override the country code when a country wants to receive broadcast video from a different national source. See the Country Codes and Channel to Frequency Mappings appendix for a list of country codes.

Previous Home Topic Contents Index Next

IAMTVTuner::put_InputType

IAMTVTuner Interface

Sets the tuner input type (cable or antenna).

HRESULT put_ InputType(
long *lIndex*,
TunerInputType *InputType*
);

Parameters

lIndex
[in] Index value that specifies the input pin to be set.
InputType
[in] Indicates the connection type, as specified in the TunerInputType data type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

IAMTVTuner::put_TuningSpace

IAMTVTuner Interface

Sets a storage index for regional channel to frequency mappings.

HRESULT put_TuningSpace(
long *lTuningSpace*
);

Parameters

lTuningSpace
[in] Value indicating the current locale.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

As TV tuners move into portable systems, you must retain locale-specific mappings of available channels and their actual frequencies. Formulating different *lTuningSpace* values for each locale provides a way of switching the channel to frequency mappings when moving from region to region.

Previous Home Topic Contents Index Next

IAMTVTuner::StoreAutoTune

IAMTVTuner Interface

Saves the fine-tuning information for all channels.

HRESULT StoreAutoTune( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

Override the channel to frequency information stored by this method by setting a new country code in the IAMTVTuner::put_CountryCode method. See the Country Codes and Channel to Frequency Mappings appendix for a listing of country codes.

Previous Home Topic Contents Index Next

IAMVfwCaptureDialogs Interface

The **IAMVfwCaptureDialogs** interface enables an application to display one of the three dialog boxes (Source, Format, or Display) provided by Microsoft® Video for Windows® capture drivers.

When to Implement

The Video for Windows VFW Video Capture filter implements this interface. It isn't expected that anything else will implement this interface.

When to Use

Any application that enables the user to change settings in a Video for Windows capture driver-supplied dialog box should use this interface.

Methods in Vtable Order

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

IAMVfwCaptureDialogs methods	Description
HasDialog	Determines if the specified dialog box exists in the driver.
ShowDialog	Displays the specified dialog box.
SendDriverMessage	Sends a driver-specific message.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVfwCaptureDialogs::HasDialog

IAMVfwCaptureDialogs Interface

Determines if the specified dialog box exists in the driver.

HRESULT HasDialog(
int *iDialog* **);**

Parameters

iDialog
[in] Desired dialog box. This is a member of the VfwCaptureDialogs enumerated data type.

Return Values

Returns S_OK if the driver contains the dialog box or S_FALSE otherwise.

Remarks

This method calls the Video for Windows videoDialog function to query for the existence of the appropriate dialog box.

Previous Home Topic Contents Index Next

IAMVfwCaptureDialogs::SendDriverMessage

IAMVfwCaptureDialogs Interface

Sends a driver-specific message.

HRESULT SendDriverMessage(
int *iDialog*,
int *uMsg*,
long *dw1*,
long *dw2* **);**

Parameters

iDialog
[in] Handle of the driver dialog box. This is a member of the VfwCaptureDialogs enumerated data type.
uMsg
[in] Message to send to the driver.
dw1
[in] Message data.
dw2
[in] Message data.

Return Values

Return value varies depending on the implementation within each driver.

Remarks

You should never need to use this method. This method can send any private message to the capture driver. Behavior might be undetermined in response to arbitrary messages; use this method at your own risk.

This method calls the Video for Windows videoMessage function to send the driver message.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVfwCaptureDialogs::ShowDialog

IAMVfwCaptureDialogs Interface

Displays the specified dialog box.

HRESULT ShowDialog(
int *iDialog*,
long *hwnd* **);**

Parameters

iDialog
[in] Dialog box to display. This is a member of the VfwCaptureDialogs enumerated data type.
hwnd
[in] Handle of the dialog box's parent window.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

You can't use this method when the driver is streaming or displaying another dialog box. While the driver displays the dialog box you can't stream (pause or run) the filter.

IAMVfwCaptureDialogs::ShowDialog calls the Video for Windows® videoDialog function to display the appropriate dialog box.

Previous Home Topic Contents Index Next

IAMVfwCompressDialogs Interface

The **IAMVfwCompressDialogs** interface enables an application to display a Video for Windows codec (compressor/decompressor) Configure or About dialog box and to set and retrieve compressor status.

When to Implement

Microsoft's video compression manager (VCM) compressor filter (AVI Compressor) implements this interface. Other filters should not need to implement it.

When to Use

An application should use this interface when it must enable the user to change compression settings in an VCM compressor's Configure dialog box or to view the compressor's About dialog box. Applications also use this interface to set and retrieve compressor status.

Methods in Vtable Order

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

IAMVfwCompressDialogs methods	Description
ShowDialog	Displays the specified dialog box.
GetState	Retrieves the current configuration settings for the VCM codec currently being used.
SetState	Sets configuration for the ICM codec.
SendDriverMessage	Sends a driver-specific message.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVfwCompressDialogs::GetState

IAMVfwCompressDialogs Interface

Retrieves the current configuration settings for the VCM codec currently being used.

HRESULT GetState(
LPVOID *pState*,
int *pcbState

Parameters

pState
[out] State of the VCM codec.
pcbState
[in, out] Size of the state.

Return Values

Return value varies depending on the implementation within each driver.

Remarks

This method calls the COM ICGetState macro.

Previous Home Topic Contents Index Next

IAMVfwCompressDialogs::SendDriverMessage

IAMVfwCompressDialogs Interface

Sends a driver-specific message.

HRESULT SendDriverMessage(
int *uMsg*,
long *dw1*,
long *dw2* **);**

Parameters

uMsg
[in] Message to send to the driver.
dw1
[in] Message data.
dw2
[in] Message data.

Return Values

Return value varies depending on the implementation within each driver.

Remarks

You should never need to use this method. This method can send any private message to the video compressor (codec). Behavior might be undetermined in response to arbitrary messages; use this method at your own risk.

This method calls the Video for Windows video compression manager (VCM) ICSendMessage function to send the message.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVfwCompressDialogs::SetState

IAMVfwCompressDialogs Interface

Sets configuration for the VCM codec.

HRESULT SetState(

LPVOID *pState*,
int *cbState* **);**

Parameters

pState
[in] State of the VCM codec.
cbState
[in] Size of the state.

Return Values

Return value varies depending on the implementation within each driver.

Remarks

This method calls the COM **ICSetState** macro, which notifies a video compression driver to set the state of the compressor.

Previous Home Topic Contents Index Next

IAMVfwCompressDialogs::ShowDialog

IAMVfwCompressDialogs Interface

Displays the specified dialog box.

HRESULT ShowDialog(
int *iDialog*,
long *hwnd* **);**

Parameters

iDialog
[in] Dialog box to display. This is a member of the VfwCompressDialogs enumerated data type.
hwnd
[in] Handle of the dialog box's parent window.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

This method returns an error when the driver is streaming or displaying another dialog box.

While the driver displays the dialog box you can't stream (pause or run) the filter.

IAMVfwCompressDialogs::ShowDialog calls the Video for Windows video compression manager (VCM) functions ICConfigure and ICAbout to display the appropriate dialog box.

IAMVideoCompression Interface

The **IAMVideoCompression** pin interface enables you to control compression parameters that aren't part of the media type.

The put_PFramesPerKeyFrame and get_PFramesPerKeyFrame methods refer to predicted (P) frames and bidirectional (B) frames, which are MPEG concepts and not generally applicable to simpler types of compressors.

When to Implement

Implement this interface on the output pin of a video capture or video compressor filter that provides compressed video data.

When to Use

An application can use this interface to control how video is compressed, including characteristics such as the number of key frames and frame quality. Use it to retrieve a textual description of the compressor and other available information, including the compressor's capabilities.

If you are using a WDM video capture or compression filter, you can only query for this interface if the capture filter is connected to another filter in the graph.

Methods in Vtable Order

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

IAMVideoCompression methods	Description
put_KeyFrameRate	Sets the key frame rate.
get_KeyFrameRate	Retrieves the key frame rate.
put_PFramesPerKeyFrame	Sets the predicted (P) frame frequency.
get_PFramesPerKeyFrame	Retrieves the P frame frequency.
put_Quality	Sets the quality of the video image compression.
get_Quality	Retrieves the current image quality setting.

put_WindowSize	Sets the number of frames over which the compressor must maintain an average data rate.
get_WindowSize	Retrieves the number of frames over which the compressor must maintain an average data rate.
GetInfo	Retrieves compressor information.
OverrideKeyFrame	Forces a particular frame to be a key frame.
OverrideFrameSize	Overrides a particular frame's data rate.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoCompression::GetInfo

IAMVideoCompression Interface

Retrieves compressor information.

HRESULT GetInfo(
WCHAR * *pszVersion*,
int *pcbVersion,
LPWSTR *pszDescription*,
int *pcbDescription,
long *pDefaultKeyFrameRate,
long *pDefaultPFramesPerKey,
double *pDefaultQuality,
long *pCapabilities
) PURE;

Parameters

pszVersion
[out] Pointer to a version string, such as "Version 2.1.0".
pcbVersion
[in,out] Size needed for a version string. Pointer to the number of bytes in the Unicode string, not the number of characters, so it must be twice the number of characters the string can hold. Call with this set to NULL to retrieve the current size.
pszDescription
[out] Pointer to a description string, such as "Awesome Video Compressor".
pcbDescription
[in,out] Size needed for a description string. Pointer to the number of bytes in the Unicode string, not the number of characters, so it must be twice the number of characters the string can hold. Call with this set to NULL to retrieve the current size.
pDefaultKeyFrameRate
[out] Pointer to receive the default key frame rate.
pDefaultPFramesPerKey

[out] Pointer to receive the default predicted (P) frames per key frame.

pDefaultQuality

[out] Pointer to receive the default quality.

pCapabilities

[out] Pointer to receive the compression capabilities, which are a combination of the CompressionCaps data type flags.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoCompression::get_KeyFrameRate

IAMVideoCompression Interface

Retrieves the current key frame rate.

```
get_KeyFrameRate(
  long * pKeyFrameRate
  ) PURE;
```

Parameters

pKeyFrameRate

[out] Compressor's current key frame rate. A negative value means it is using the default frame rate for the video compressor. A zero value means only the first frame is a key frame.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

To determine if the compressor supports this method, check for the CompressionCaps_CanKeyFrame flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

See Also

IAMVideoCompression::put_KeyFrameRate

IAMVideoCompression::get_PFramesPerKeyFram

IAMVideoCompression Interface

Retrieves the predicted (P) frame interval.

HRESULT get_PFramesPerKeyFrame(
long * *pPFramesPerKeyFrame*
) PURE;

Parameters

pPFramesPerKeyFrame
[out] Pointer to receive the number of P frames per key frame. A negative value means the compressor will use its default value.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The Video for Windows capture filter (VFW Video Capture) and the AVI compression filter (AVI Compressor) do not currently support this interface and return E_NOTIMPL.

Remarks

To determine if the compressor supports this method, check for the CompressionCaps_CanBFrame flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

As an example of the relationship between the types of frames, suppose a key frame occurs once in every 10 frames, and there are 3 P frames per key frame. The P frames will be spaced evenly between the key frames. The other 6 frames, which occur between the key frames and the P frames, will be bidirectional (B) frames.

See Also

IAMVideoCompression::put_KeyFrameRate

IAMVideoCompression::get_Quality

IAMVideoCompression Interface

Retrieves the current image quality setting.

```
HRESULT get_Quality(
  double * pQuality
  ) PURE;
```

Parameters

pQuality
[out] Current quality setting.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

The quality is a value between 0 and 1. One indicates the highest (best) quality and 0 indicates the lowest (worst) quality. A negative number means it is using the compressor default. The compressor interprets this number; this interpretation varies from compressor to compressor. When the compressor is not compressing to a specific data rate, the value will roughly determine the image size or quality.

To determine if the compressor supports this method, check for the CompressionCaps_CanQuality flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

If you are compressing to a fixed data rate, a high quality value means use all of the data rate, and a low quality value means you can use much lower than the data rate.

Previous Home Topic Contents Index Next

IAMVideoCompression::get_WindowSize

IAMVideoCompression Interface

Retrieves the number of frames over which the compressor must maintain an average data rate.

```
HRESULT get_WindowSize(
  DWORDLONG * pWindowSize
  );
```

Parameters

pWindowSize
[out] Pointer to a DWORDLONG value that will receive the window size.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

See Also

IAMVideoCompression::put_WindowSize

Previous Home Topic Contents Index Next

IAMVideoCompression::OverrideFrameSize

IAMVideoCompression Interface

Overrides a frame's data rate.

```
HRESULT OverrideFrameSize(
  long FrameNumber,
  long Size
  ) PURE;
```

Parameters

FrameNumber
[in] Frame number for which to change the size.
Size
[in] Desired size, in bytes, for the specified frame.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The Video for Windows capture filter (VFW Video Capture) and the AVI compression filter (AVI Compressor) do not currently support this interface and return E_NOTIMPL.

Remarks

To determine if the compressor might support this method, check for the CompressionCaps_CanCrunch flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method. The flag might also be set to indicate that the dwBitRate value can be set in the AM_MEDIA_TYPE's VIDEOINFOHEADER structure.

The frame number refers to which frame goes out of the filter after it is streaming. For example, frame 0 means the first frame this pin delivers. Frame 11 means the twelfth frame it delivers. Be sure to call this method before the filter delivers the frame for which you want to provide a different size.

Overriding the frame size (or "crunching" the frame) instructs the filter to make the frame size this many bytes or less instead of the originally planned size.

Previous Home Topic Contents Index Next

IAMVideoCompression::OverrideKeyFrame

IAMVideoCompression Interface

Forces a frame to be a key frame.

HRESULT OverrideKeyFrame(
long *FrameNumber*
) PURE;

Parameters

FrameNumber
[in] Number of the frame to be made a key frame when the graph runs, even if it wouldn't usually be a key frame.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The Video for Windows capture filter (VFW Video Capture) and the AVI compression filter (AVI Compressor) do not currently support this interface and return E_NOTIMPL.

Remarks

Once a compressor creates a key frame, it might reset its count to determine when the next key frame should occur. For example, assume a key frame typically occurs once every 10 frames. If you mark frame 5 as a key frame using **OverrideKeyFrame**, the compressor might wait 10 more frames until creating the next key frame.

Previous Home Topic Contents Index Next

IAMVideoCompression::put_KeyFrameRate

IAMVideoCompression Interface

Sets the key frame rate.

HRESULT put_KeyFrameRate(
long *KeyFrameRate*
) PURE;

Parameters

KeyFrameRate
[in] Desired key frame rate. A negative value means use the default frame rate for the video compressor. A zero value means that only the first frame is a key frame.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

To determine if the compressor supports this method, check for the CompressionCaps_CanKeyFrame flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

Previous Home Topic Contents Index Next

IAMVideoCompression::put_PFramesPerKeyFram

IAMVideoCompression Interface

Sets predicted (P) frame interval.

HRESULT put_PFramesPerKeyFrame(
long *PFramesPerKeyFrame*
) PURE;

Parameters

PFramesPerKeyFrame
[in] Desired P frame interval. A negative value means use the default frame rate for the video compressor.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The Video for Windows capture filter (VFW Video Capture) and the AVI compression filter (AVI Compressor) do not currently support this interface and return E_NOTIMPL.

Remarks

To determine if the compressor supports this method, check for the CompressionCaps_CanBFrame flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

As an example of the relationship between the types of frames, suppose a key frame occurs once in every 10 frames, and there are 3 P frames per key frame. The P frames will be spaced evenly between the key frames. The other 6 frames, which occur between the key frames and the P frames, will be bidirectional (B) frames.

Previous Home Topic Contents Index Next

IAMVideoCompression::put_Quality

IAMVideoCompression Interface

Sets the quality of the video image.

HRESULT put_Quality(
double *Quality*
) PURE;

Parameters

Quality
[in] Desired quality.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

The quality is a value between 0 and 1, inclusive. One indicates the highest (best) quality and 0 indicates the lowest (worst) quality. A negative number means use the compressor default. The compressor (codec) interprets this number; interpretation varies from codec to codec. When the compressor is not compressing to a specific data rate, the value will roughly determine the image size or quality.

To determine if the compressor supports this method, check the CompressionCaps_CanQuality flag returned in the *pCapabilities* parameter of the IAMVideoCompression::GetInfo method.

If you are compressing to a fixed data rate, a high quality value means use all of the data rate, and a low quality value means you can use much lower than the data rate.

IAMVideoCompression::put_WindowSize

IAMVideoCompression Interface

Sets the number of frames over which the compressor must maintain an average data rate.

HRESULT put_WindowSize(
DWORDLONG *WindowSize*
);

Parameters

WindowSize
[in] Window size, or number of frames, whose average size cannot exceed the data rate that the compressor has been asked to provide.

Return Values

Returns an HRESULT value that depends on the implementation of the interface.

Remarks

For a window of size *n*, the average frame size of any consecutive *n* frames will not exceed the stream's specified data rate, although individual frames can be larger or smaller. For example, if you have set a data rate of 100 kilobytes (KB) per second on a 10 frames per second (fps) movie, that will usually mean each frame must be less than or equal to 10 KB. However, by setting a window size of *n*, you are specifying that as long as the average length of those *n* frames is less than or equal to 10 KB, it doesn't matter how large the individual frames are. For example, some could be smaller and some could actually be larger than 10 KB, as long as the average is less than or equal to 10 KB.

See Also

IAMVideoCompression::get_WindowSize

IAMVideoCutListElement Interface

IAMVideoCutListElement provides support for a cutlist element from an AVI video file stream.

See About Cutlists and Using Cutlists for more information.

When to Implement

Usually, you don't need to implement this interface because DirectShow provides the CLSID_VideoFileClip object that implements it for you. Implement this interface in your application when you need to change the default behavior of this interface to include support for interlaced video.

When to Use

Use this interface in your filter when you specify a video-based media clip. Call QueryInterface on the IAMCutListElement interface to determine if the element is a video type element.

When compiling a cutlist application you must explicitly include the cutlist header file as follows:

```
#include <cutlist.h>
```

Methods in Vtable Order

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

IAMVideoCutListElement methods	Description
IsSingleFrame	Determines if the element is a single frame with repeating fields.
GetStreamIndex	Retrieves the index to the specified stream in the AVI file.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoCutListElement::GetStreamIndex

IAMVideoCutListElement Interface

Retrieves the index to the specified stream in the AVI file.

HRESULT GetStreamIndex(
DWORD **piStream*
);

Parameters

piStream
[out] Pointer to the stream number to open.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_POINTER	Null pointer argument.
S_OK	Success.

Remarks

The stream number must always be zero. The only supported video stream in an AVI file is the first video stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoCutListElement::IsSingleFrame

IAMVideoCutListElement Interface

Determines if the element is a single frame with repeating fields.

HRESULT IsSingleFrame(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_POINTER	Null pointer argument.
S_FALSE	No, element is not a single frame with repeating fields.
S_OK	Yes, element is a single frame with repeating fields.

Remarks

This method must always return S_FALSE because repeating fields are not supported.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoProcAmp Interface

The **IAMVideoProcAmp** interface contains methods for controlling video quality such as brightness, contrast, hue, saturation, gamma, and sharpness. It defines a uniform range for these settings regardless of whether the adjustment is made in the analog or digital domain.

For analog video, this interface will typically be located on the same processing element as the **IAMAnalogVideoDecoder** interface.

When to Implement

Implement this interface when your filter needs to control video quality.

When to Use

Use this interface when your application needs to adjust video quality.

Methods in Vtable Order

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

IAMVideoProcAmp methods	Description
GetRange	Retrieves minimum, maximum, and default values for setting properties.
Set	Sets video quality for a specified property.
Get	Retrieves video quality for a specified property.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoProcAmp::Get

IAMVideoProcAmp Interface

Retrieves video quality for a specified property.

```
HRESULT Get(
  long Property,
  long * lValue,
  long * Flags );
```

Parameters

Property
[in] Specific property to retrieve the setting of. Specify a member of the VideoProcAmpProperty enumerated type.

lValue
[out] Current value of the property.

Flags
[out] Pointer to a member of the VideoProcAmpFlags enumerated type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method is not supported.
NOERROR	No error.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoProcAmp::GetRange

IAMVideoProcAmp Interface

Retrieves minimum, maximum, and default values for setting properties.

HRESULT GetRange(
  long *Property*,
  long * *pMin*,
  long * *pMax*,
  long * *pSteppingDelta*,
  long * *pDefault*,
  long * *pCapsFlags* **);**

Parameters

Property
[in] Specific property to determine the range of. Specify a member of the VideoProcAmpProperty enumerated type.
pMin
[out] Minimum setting range.
pMax
[out] Maximum setting range.
pSteppingDelta
[out] Step size.
pDefault
[out] Default value.
pCapsFlags
[out] Pointer to a member of the VideoProcAmpFlags enumerated type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method is not supported.
NOERROR	No error.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAMVideoProcAmp::Set

IAMVideoProcAmp Interface

Sets video quality for a specified property.

HRESULT Set(
long *Property*,
long *lValue*,
long *Flags* **);**

Parameters

Property
[in] Specific property to set. Specify a member of the VideoProcAmpProperty enumerated type.
lValue
[in] Value indicating the setting of the property.
Flags
[in] Member of the VideoProcAmpFlags enumerated type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. **HRESULT** can include one of the following standard constants, or other values not listed.

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method is not supported.
NOERROR	No error.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader Interface

The **IAsyncReader** interface allows multiple overlapped reads from different positions in the media stream. This interface is supported by source filters.

Note that during connection an output pin supporting the **IAsyncReader** should check whether its QueryInterface method is called asking for the **IAsyncReader** interface. If it is not, then the output pin should fail the connect unless it establishes some other transport to use during the connection.

When to Implement

Implement this interface on a pin if your filter reads data of media type MEDIATYPE_Stream from some source.

When to Use

A parser, such as an Apple® QuickTime® parser filter, can use this interface to read from a filter that reads from a file, the network, or memory.

Methods in Vtable Order

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

IAsyncReader methods	Description
RequestAllocator	Retrieves the actual allocator to be used.
Request	Queues a request for data.
WaitForNext	Blocks until the next sample is completed or the time-out occurs.
SyncReadAligned	Performs an aligned synchronized read.
SyncRead	Performs a synchronized read.
Length	Retrieves the total length of the stream, and the currently available length.
BeginFlush	Causes all outstanding reads to return.
EndFlush	Ends the flushing operation.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader::BeginFlush

IAsyncReader Interface

Starts the flushing operation.

HRESULT BeginFlush(void);

Return Values

Returns S_OK if successful, S_FALSE otherwise.

Remarks

Causes all outstanding reads to return, possibly with a failure code (VFW_E_TIMEOUT), indicating that the outstanding reads were canceled. Between **IAsyncReader::BeginFlush** and IAsyncReader::EndFlush calls, IAsyncReader::Request calls will fail and IAsyncReader::WaitForNext calls will always complete immediately.

Previous Home Topic Contents Index Next

IAsyncReader::EndFlush

IAsyncReader Interface

Completes the flushing operation.

HRESULT EndFlush(void);

Return Values

Returns S_OK if successful, S_FALSE otherwise.

Remarks

Between IAsyncReader::BeginFlush and **IAsyncReader::EndFlush** calls, IAsyncReader::Request calls will fail and IAsyncReader::WaitForNext calls will always complete immediately. This method is called so the source thread can wait in the

IAsyncReader::WaitForNext method again.

Previous Home Topic Contents Index Next

IAsyncReader::Length

IAsyncReader Interface

Retrieves the stream's total length, and the currently available length.

HRESULT Length(
LONGLONG* *pTotal*,
LONGLONG* *pAvailable*
);

Parameters

pTotal
Total allocated length.
pAvailable
Available length.

Return Values

Returns S_OK if successful, E_UNEXPECTED if the file has not been opened.

Remarks

Read operations beyond the available length but within the total length will normally succeed, but they might block for a long period of time.

Previous Home Topic Contents Index Next

IAsyncReader::Request

IAsyncReader Interface

Queues a request for data.

```
HRESULT Request(
  IMediaSample* pSample,
  DWORD dwUser
  );
```

Parameters

pSample
Media sample being requested.
dwUser
[in] User context.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Current DirectShow implementation return values include:

Value	Meaning
VFW_E_BADALIGN	An invalid alignment was specified.
VFW_E_MEDIA_TIME_NOT_SET	Time has not been set.
HRESULT_FROM_WIN32	Request for data past end of file.
NOERROR	No error.
S_OK	Success.

Remarks

Media sample start and stop times contain the requested absolute byte position (start-inclusive and stop-exclusive). This method might fail if the sample is not obtained from an agreed allocator or if the start or stop position does not match the agreed alignment. The samples allocated from the source pin's allocator might fail IMediaSample::GetPointer until after returning from IAsyncReader::WaitForNext.

The stop position must be aligned, which means it might exceed duration. On completion, the stop position will be corrected to the unaligned actual data.

The *dwUser* parameter is used by the caller to identify the sample that returned from the IAsyncReader::WaitForNext method. It has no meaning within IAsyncReader but could be used to track individual sample information.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader::RequestAllocator

IAsyncReader Interface

Retrieves the actual allocator to be used.

HRESULT RequestAllocator(
IMemAllocator* *pPreferred*,
ALLOCATOR_PROPERTIES* *pProps*,
IMemAllocator ** *ppActual*
);

Parameters

pPreferred
[in] Preferred allocator.
pProps
[in] Preferred allocator properties (size, count, and alignment).
ppActual
[out] Actual allocator used.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Current DirectShow implementation return values include:

Value	Meaning
E_FAIL	Failure to initialize an allocator.
VFW_E_BADALIGN	An invalid alignment was specified.
S_OK	Allocator was returned.

Remarks

The preferred allocator and preferred allocator properties must be passed in. This method returns the actual allocator to be used.

IMemAllocator::GetProperties should be called on the returned allocator to learn the alignment and prefix chosen. This allocator will not be committed and decommitted by the asynchronous reader, only by the consumer. This method must be called before calling IAsyncReader::Request.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader::SyncRead

IAsyncReader Interface

Performs a synchronous read.

HRESULT SyncRead(
LONGLONG *llPosition*,
LONG *lLength*,
BYTE* *pBuffer*
);

Parameters

llPosition
[in] Absolute file position.
lLength
[in] Number of bytes required.
pBuffer
[out] Where the data is written.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Current DirectShow implementation return values include:

Value	Meaning
VFW_E_BADALIGN	An invalid alignment was specified.
HRESULT_FROM_WIN32	Win32 error.
S_FALSE	Size changed (probably due to end of file).
S_OK	Success.

Remarks

The **SyncRead** method works in a stopped state as well as in a running state. The read is not necessarily aligned. This method fails if the read is beyond the actual total length.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader::SyncReadAligned

IAsyncReader Interface

Performs a synchronous read of the data.

HRESULT SyncReadAligned(
IMediaSample* *pSample*

);

Parameters

pSample
Sample to read.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Current DirectShow implementation return values include:

Value	Meaning
VFW_E_BADALIGN	An invalid alignment was specified.
HRESULT_FROM_WIN32	Win32 error.
S_FALSE	Size changed (probably due to end of file).
S_OK	Success.

Remarks

The sample passed in must have been acquired from the agreed allocator. The start and stop positions must be aligned equivalent to an IAsyncReader::Request/IAsyncReader::WaitForNext pair, but may avoid the need for a thread on the source filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IAsyncReader::WaitForNext

IAsyncReader Interface

Blocks until the next read requested through IAsyncReader::Request completes or the time-out occurs.

HRESULT WaitForNext(
DWORD *dwTimeout*,
IMediaSample** *ppSample*,
DWORD * *pdwUser*
);

Parameters

dwTimeout
[in] Time-out in milliseconds; can be zero or INFINITE.
ppSample

[out] Completed sample.

pdwUser

User context.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Current DirectShow implementation return values include:

Value	Meaning
VFW_E_TIMEOUT	A time-out has expired.
VFW_E_WRONG_STATE	The operation could not be performed because the filter is in the wrong state.
E_FAIL	Failure.
S_OK	Success.

Remarks

Samples may not be returned in order. If there is a read error of any sort, a notification will already have been sent by the source filter, and HRESULT will be an error. If *ppSample* is not null, a request has been completed with the result code returned.

The *pdwUser* parameter returns the caller's context DWORD corresponding to the sample returned.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter Interface

The **IBaseFilter** interface abstracts an object that has typed input and output connections and can be aggregated dynamically. All DirectShow™ filters expose this interface.

Since the **IBaseFilter** interface derives from the IMediaFilter interface, it inherits IPersist.

When to Implement

Implement this interface on every DirectShow filter. It is recommended that you use the CBaseFilter class library to implement this interface.

When to Use

The filter graph manager is the primary user of this interface. Applications or other filters can use IBaseFilter methods directly to enumerate or retrieve pins or to get vendor information, but should not use any methods derived from IMediaFilter to control media streaming (use the

IMediaControl methods on the filter graph manager instead).

Methods in Vtable Order

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

IMediaFilter methods	Description
Stop	Informs the filter to transition to the new (stopped) state.
Pause	Informs the filter to transition to the new (paused) state.
Run	Informs the filter to transition to the new (running) state.
GetState	Determines the state of the filter.
SetSyncSource	Identifies the reference clock to which the filter should synchronize activity.
GetSyncSource	Retrieves the current reference clock (or NULL if there is no clock). Passes a time value to synchronize independent streams.

IBaseFilter methods	Description
EnumPins	Enumerates the specified pins available on this filter.
FindPin	Retrieves a pointer to the pin with the specified identifier.
QueryFilterInfo	Retrieves information about the specified filter.
JoinFilterGraph	Notifies a filter that it has joined a filter graph.
QueryVendorInfo	Retrieves optional information supplied by a vendor for the specified filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter::EnumPins

IBaseFilter Interface

Enumerates all the pins available on this filter.

```
HRESULT EnumPins(
  IEnumPins ** ppEnum
  );
```

Parameters

ppEnum
[out] Pointer to the IEnumPins interface to retrieve.

Return Values

Returns an HRESULT value that depends on the implementation. **HRESULT** can be one of the following standard constants, or other values not listed:

Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method isn't supported.
S_OK or NOERROR	Success.

Remarks

The interface returned by this method has had its reference count incremented. Be sure to use IUnknown::Release on the interface to decrement the reference count when you have finished using the interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter::FindPin

IBaseFilter Interface

Retrieves the pin with the specified identifier.

```
HRESULT FindPin(
  LPCWSTR Id,
  IPin **ppPin
  );
```

Parameters

Id
[in] Identifier of the pin.
ppPin
[out] Pointer to the IPin interface for this pin after the filter has been restored. The returned **IPin** pointer has been reference counted. The caller should use the Release method on the pointer when finished with it.