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, along with the IPin::QueryId method, is used to implement persistent filter graphs. A filter must be able to translate the IPin interface pointers to its pins into identifiers that can be saved along with the configuration of the filter graph. It does this by using the **IPin::QueryId** method. It must then be able to convert those identifiers back into **IPin** interface pointers when the filter and its connections are restored as part of a persistent filter graph. This is accomplished by using the **IBaseFilter::FindPin** method.

The *ppPin* parameter is set to NULL if the identifier cannot be matched.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter::JoinFilterGraph

IBaseFilter Interface

Notifies a filter that it has joined a filter graph.

```
HRESULT JoinFilterGraph(
  IFilterGraph * pGraph,
  LPCWSTR pName
  );
```

Parameters

pGraph
[in] Pointer to the filter graph to join.

pName
[in, string] Name of the filter being added.

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 filter should store this interface for later use because it might need to notify the interface about events. The filter should not add a reference count to this interface. Doing so causes circular references and prevents the graph or the filter from ever being correctly released. The filter graph manager does hold a reference on the filter. The filter does not need to hold a reference to ensure that the interface does not disappear; it is notified of any such disappearance by a further call to **IBaseFilter::JoinFilterGraph**, with a null value for the *pGraph* parameter to indicate that the filter is no longer part of the graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter::QueryFilterInfo

IBaseFilter Interface

Returns information about the filter.

```
HRESULT QueryFilterInfo(
  FILTER_INFO * pInfo
  );
```

Parameters

pInfo
[out] Pointer to a FILTER_INFO structure.

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 FILTER_INFO structure is defined as follows:

```
typedef struct _FilterInfo  {
    [string] WCHAR achName[MAX_FILTER_NAME]; // allowed to be null
    IFilterGraph * pGraph;                   // null if not part of graph
    } FILTER_INFO ;
```

Note that on return, if the pGraph member of the FILTER_INFO structure is non-NULL, it will have an outstanding reference count and should be released when the interface is no longer needed.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBaseFilter::QueryVendorInfo

IBaseFilter Interface

Returns a vendor information string.

HRESULT QueryVendorInfo(
LPWSTR * *pVendorInfo*
);

Parameters

pVendorInfo
[out] Pointer to a string containing vendor information.

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 is optional and can return E_NOTIMPL. If implemented, memory returned should be freed using the Microsoft® Win32® CoTaskMemFree function when the application is done using it.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicAudio Interface

IBasicAudio is an interface that supports the filter graph's audio component. It allows access to volume and balance functionality.

The Volume property is a value between –10,000 and 0 representing a set of logarithmic steps. This follows the DirectSound model. Not all devices support 10,000 distinguishable steps.

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

When to Implement

The audio renderer filter supplied with Microsoft® DirectShow™ implements this interface. It is also implemented by the filter graph manager (by means of a plug-in distributor) to pass method calls from the application to the audio renderer filter's implementation of the interface.

Implement this interface if you are writing a replacement audio renderer filter or a replacement DirectShow plug-in distributor. You can use the CBasicAudio class, which handles the IDispatch implementation for Automation, to help implement this interface.

When to Use

When the filter graph manager exposes this interface, it is used by applications that need to control the properties of the audio renderer filter. Applications should not use the interface

exposed by the audio renderer directly. When the audio renderer filter exposes this interface, it is used by plug-in distributors, such as the DirectShow plug-in distributor, which pass calls from the application.

Methods in Vtable Order

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

IDispatch methods	Description
GetTypeInfoCount	Determines whether there is type information available for this dispinterface.
GetTypeInfo	Retrieves the type information for this dispinterface if GetTypeInfoCount returned successfully.
GetIDsOfNames	Converts text names of properties and methods (including arguments) to their corresponding DISPIDs.
Invoke	Calls a method or accesses a property in this dispinterface if given a DISPID and any other necessary parameters.

IBasicAudio methods	Description
put_Volume	Sets the volume (amplitude) of the audio signal.
get_Volume	Retrieves the volume (amplitude) of the audio signal.
put_Balance	Sets the balance for the audio signal.
get_Balance	Retrieves the balance for the audio signal.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicAudio::get_Balance

IBasicAudio Interface

Retrieves the balance for the audio signal.

```
HRESULT get_Balance(
  long * plBalance
  );
```

Parameters

plBalance
[out] Returned value of the Balance property.

Return Values

Returns an HRESULT value.

Remarks

As with the Volume property, units correspond to .01 decibels (multiplied by –1 when *plBalance* is a positive value). For example, a value of 1000 indicates –10 dB on the right channel and –90 dB on the left channel.

Previous Home Topic Contents Index Next

IBasicAudio::get_Volume

IBasicAudio Interface

Retrieves the volume (amplitude) of the audio signal.

HRESULT get_Volume(
long * *plVolume*
);

Parameters

plVolume
[out] Returned value of the Volume property. Divide by 100 to get equivalent decibel value (for example –10,000 = –100 dB).

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicAudio::put_Balance

IBasicAudio Interface

Sets the balance of the audio signal.

HRESULT put_Balance(
long *lBalance*
);

Parameters

lBalance
[in] Value to which to set the Balance property. The allowable input range is –10,000 to 10,000. A value of 0 sets a neutral balance—both left and right speakers will be given the same amplitude audio signal.

Return Values

Returns an HRESULT value. E_INVALIDARG is returned for values outside the allowable input range and E_FAIL is returned if the underlying device returns an error.

Remarks

As with the Volume property, units correspond to .01 decibels (multiplied by –1 when *lBalance* is a positive value). For example, a value of 1000 indicates –10 dB on the right channel and –90 dB on the left channel.

Previous Home Topic Contents Index Next

IBasicAudio::put_Volume

IBasicAudio Interface

Sets the volume (amplitude) of the audio signal.

HRESULT put_Volume(
long *lVolume*
);

Parameters

lVolume
[in] Value to which to set the Volume property. The allowable input range is –10,000 to 0.

Return Values

Returns an HRESULT value. E_INVALIDARG is returned for values outside the allowable input

range and E_FAIL is returned if the underlying device returns an error.

Remarks

Full volume is 0, and –10,000 is silence; the scale is logarithmic. Multiply the desired decibel level by 100 (for example –10,000 = –100 dB).

IBasicVideo Interface

The **IBasicVideo** interface supports the video properties of a generic video window. Generally, this is a video renderer that draws video into a window on the display.

The **IBasicVideo** interface supports both properties and methods. Properties are more easily accessible from many Automation controllers (such as the Microsoft® Visual Basic® programming system). However, some operations require several properties to be changed simultaneously; for this reason, methods are provided that allow a number of related properties to be changed.

The **IBasicVideo** interface methods require only that the video renderer be connected. If it is not connected, all the interface methods return VFW_E_NOT_CONNECTED. Properties set on a video renderer persist between successive connections and disconnections. All applications should ensure that they reset the renderer properties before starting a presentation.

When working with video, the application can select a portion of the video to use. This portion is the source rectangle that the **IBasicVideo** interface controls. **IBasicVideo** allows the source rectangle to be set and retrieved. All the rectangles that **IBasicVideo** uses employ top, left, width, and height rather than top, left, right, and bottom, which is favored in Microsoft Win32® programming. When no source rectangle has been set, the properties of the source rectangle return the full, native video size.

When to Implement

The video renderer filter supplied with Microsoft DirectShow™ implements this interface. It is also implemented by the filter graph manager (via a plug-in distributor) to pass method calls from the application to the video renderer filter's implementation of the interface. Implement this interface if you are writing a replacement video renderer filter or a replacement DirectShow plug-in distributor. You can use the CBaseBasicVideo class, which handles the IDispatch implementation for Automation, to help implement this interface.

When to Use

When the filter graph manager exposes this interface, it is used by applications that must control the properties of the video renderer filter.

Methods in Vtable Order

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

IDispatch methods	Description
GetTypeInfoCount	Determines whether there is type information available for this dispinterface.
GetTypeInfo	Retrieves the type information for this dispinterface if GetTypeInfoCount returned successfully.
GetIDsOfNames	Converts text names of properties and methods (including arguments) to their corresponding DISPIDs.
Invoke	Calls a method or accesses a property in this dispinterface if given a DISPID and any other necessary parameters.

IBasicVideo methods	Description
get_AvgTimePerFrame	Retrieves the average time between successive frames in 100-nanosecond units.
get_BitRate	Retrieves an approximate bit rate for the video stream.
get_BitErrorRate	Retrieves an approximate bit error rate for the video stream.
get_VideoWidth	Retrieves the current video width.
get_VideoHeight	Retrieves the current video height.
put_SourceLeft	Sets the x-axis coordinate for the source video rectangle.
get_SourceLeft	Retrieves the x-axis coordinate for the source video rectangle.
put_SourceWidth	Sets the width of the source video rectangle.
get_SourceWidth	Retrieves the width of the source video rectangle.
put_SourceTop	Sets the y-axis coordinate for the source video rectangle.
get_SourceTop	Retrieves the y-axis coordinate for the source video rectangle.
put_SourceHeight	Sets the height of the source video rectangle.
get_SourceHeight	Retrieves the height of the source video rectangle.
put_DestinationLeft	Sets the x-axis coordinate for the destination video rectangle.
get_DestinationLeft	Retrieves the x-axis coordinate for the destination video rectangle.
put_DestinationWidth	Sets the width of the destination video rectangle.
get_DestinationWidth	Retrieves the width of the destination video rectangle.
put_DestinationTop	Sets the y-axis coordinate for the destination video rectangle.
get_DestinationTop	Retrieves the y-axis coordinate for the destination video rectangle.
put_DestinationHeight	Sets the height of the destination video rectangle.
get_DestinationHeight	Retrieves the height of the destination video rectangle.
SetSourcePosition	Sets the source video rectangle.
GetSourcePosition	Retrieves the source video rectangle.
SetDefaultSourcePosition	Informs the renderer to use the default source rectangle.
SetDestinationPosition	Sets the destination rectangle for the window.
GetDestinationPosition	Retrieves the destination video rectangle for the window.
SetDefaultDestinationPosition	Sets the default destination position for the window.
GetVideoSize	Retrieves the native video dimensions.

GetVideoPaletteEntries	Retrieves the color palette entries required by the video.
GetCurrentImage	Returns a copy of the current image that is waiting at the renderer.
IsUsingDefaultSource	Determines if the renderer is using the default source rectangle.
IsUsingDefaultDestination	Determines if the renderer is using the default destination rectangle.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicVideo::get_AvgTimePerFrame

IBasicVideo Interface

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

HRESULT get_AvgTimePerFrame(
REFTIME *pAvgTimePerFrame*
);

Parameters

pAvgTimePerFrame
[out] Time between frames.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::get_BitErrorRate

IBasicVideo Interface

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

HRESULT get_BitErrorRate(
long **pBitErrorRate*
);

Parameters

pBitErrorRate
[out] Approximate bit error rate.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::get_BitRate

IBasicVideo Interface

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

HRESULT get_BitRate(
long **pBitRate*
);

Parameters

pBitRate
[out] Approximate bit rate.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::GetCurrentImage

IBasicVideo Interface

Retrieves the current image waiting at the renderer.

HRESULT GetCurrentImage(
long **pBufferSize*,
long **pDIBImage*
);

Parameters

pBufferSize
[in, out] Size of the buffer the caller is passing in.
pDIBImage
[out] Pointer to a buffer where the complete image will be stored in device-independent bitmap (DIB) format. The buffer must be cast to a long pointer for the function.

Return Values

Returns an HRESULT value.

Remarks

An application can use this method to get a copy of the current image the video renderer holds when paused. The size of the buffer required to hold the image can be obtained by calling the method with a null image pointer. In this case, the buffer size is filled out with the byte count required. The buffer size pointer must always be valid. If the application calls this method and the buffer size is too small to hold the complete image, the renderer returns E_OUTOFMEMORY.

Pause the video renderer before calling this method. Calling this method in any other state than paused returns VFW_E_NOT_PAUSED. Depending on what data the source filter has available, the video renderer is not guaranteed to service this request. If no image is available, it returns E_FAIL.

When called successfully, the output image pointer is filled with the entire DIB image, including the Microsoft Win32 BITMAPINFOHEADER structure, any required palette, and bit masks as defined in the Win32 BITMAPINFO structure. The format of the image that is returned depends on the type provided by the source filter, and cannot be explicitly defined in advance. Typically, the image will be a 24-bit, 16-bit, or 8-bit palettized image.

Previous Home Topic Contents Index Next

IBasicVideo::get_DestinationHeight

IBasicVideo Interface

Retrieves the height of the destination rectangle.

HRESULT get_DestinationHeight(
long **pDestinationHeight*
);

Parameters

pDestinationHeight
[out] Current height.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::get_DestinationLeft

IBasicVideo Interface

Retrieves the destination x-axis origin coordinate.

HRESULT get_DestinationLeft(
long **pDestinationLeft*
);

Parameters

pDestinationLeft
[out] Current x-axis origin.

Return Values

Returns an HRESULT value.

Remarks

Setting this coordinate does not affect the destination rectangle width.

Previous Home Topic Contents Index Next

IBasicVideo::GetDestinationPosition

IBasicVideo Interface

Retrieves the position of the destination rectangle in window coordinates.

HRESULT GetDestinationPosition(
long *pLeft*,
long *pTop*,
long *pWidth*,
long *pHeight*
);

Parameters

pLeft
[out] The x-axis origin of the destination window.
pTop
[out] The y-axis origin of the destination window.
pWidth
[out] Width of the destination window.
pHeight
[out] Height of the destination window.

Return Values

Returns an HRESULT value.

Remarks

This method has the same effect as individually calling the IBasicVideo::get_DestinationLeft, IBasicVideo::get_DestinationTop, IBasicVideo::get_DestinationWidth, and IBasicVideo::get_DestinationHeight methods.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicVideo::get_DestinationTop

IBasicVideo Interface

Retrieves the destination y-axis origin coordinate.

HRESULT get_DestinationTop(
long **pDestinationTop*
);

Parameters

pDestinationTop
[out] Current y-axis origin.

Return Values

Returns an HRESULT value.

Remarks

Setting this coordinate does not affect the destination rectangle height.

Previous Home Topic Contents Index Next

IBasicVideo::get_DestinationWidth

IBasicVideo Interface

Retrieves the width of the destination rectangle.

HRESULT get_DestinationWidth(
long **pDestinationWidth*
);

Parameters

pDestinationWidth
[out] Current width.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::get_SourceHeight

IBasicVideo Interface

Retrieves the height of the source rectangle.

HRESULT get_SourceHeight(
long **pSourceHeight*
);

Parameters

pSourceHeight
[out] Current rectangle height.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::get_SourceLeft

IBasicVideo Interface

Retrieves the source rectangle x-axis coordinate.

HRESULT get_SourceLeft(
long **pSourceLeft*
);

Parameters

pSourceLeft
[out] The x-axis origin of the source rectangle.

Return Values

Returns an HRESULT value.

Remarks

This method has no effect on the source rectangle width.

Previous Home Topic Contents Index Next

IBasicVideo::GetSourcePosition

IBasicVideo Interface

Retrieves the source position rectangle, clipped to the available video.

HRESULT GetSourcePosition(
long *pLeft*,
long *pTop*,
long *pWidth*,
long *pHeight*
);

Parameters

pLeft
[out] The x-axis origin of the source window.
pTop
[out] The y-axis origin of the source window.
pWidth
[out] Width of the source window.
pHeight
[out] Height of the source window.

Return Values

Returns an HRESULT value.

Remarks

This method has the same effect as individually calling the IBasicVideo::get_SourceLeft, IBasicVideo::get_SourceTop, IBasicVideo::get_SourceWidth, and IBasicVideo::get_SourceHeight methods.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicVideo::get_SourceTop

IBasicVideo Interface

Retrieves the source rectangle y-axis origin coordinate.

HRESULT get_SourceTop(
long * *pSourceTop*
);

Parameters

pSourceTop
[out] New top coordinate.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::get_SourceWidth

IBasicVideo Interface

Retrieves the source rectangle width.

HRESULT get_SourceWidth(
long *pSourceWidth*
);

Parameters

pSourceWidth
[out] Current width.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::get_VideoHeight

IBasicVideo Interface

Retrieves the current video height.

```
HRESULT get_VideoHeight(
  long *pVideoHeight
  );
```

Parameters

pVideoHeight
[out] Native height of the video.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::GetVideoPaletteEntries

IBasicVideo Interface

Retrieves the palette colors for the video.

HRESULT GetVideoPaletteEntries(
long *StartIndex*,
long *Entries*,
long **pRetrieved*,
long **pPalette*
);

Parameters

StartIndex
[in] Start index for the palette.
Entries
[in] Number of palette entries required.
pRetrieved
[out] Number of entries actually returned.
pPalette
[out] Pointer to an array of Microsoft Win32 PALETTEENTRY structures.

Return Values

Returns an HRESULT value.

Remarks

An application can query the colors required by the video with this method. An application can pass in a null *pPalette* parameter, and the *pRetrieved* parameter is filled in with the number of colors in use. The *pPalette* parameter is a pointer to an array of PALETTEENTRY structures. With the *StartIndex* and *Entries* parameters, an application can request any contiguous section of the palette in a single method. The interface was modeled on the Win32 GetSystemPaletteEntries function.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicVideo::GetVideoSize

IBasicVideo Interface

Retrieves the native video dimensions.

HRESULT GetVideoSize(

long **pWidth*,
long **pHeight*
);

Parameters

pWidth
[out] Width of the video window.
pHeight
[out] Height of the video window.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::get_VideoWidth

IBasicVideo Interface

Retrieves the current video width.

HRESULT get_VideoWidth(
long **pVideoWidth*
);

Parameters

pVideoWidth
[out] Native width of the video.

Return Values

Returns an HRESULT value.

Remarks

This method returns the actual width of the video supplied to the renderer. It does not address any set source rectangle, but simply returns the native horizontal dimension. Applications can

use it to negotiate size requirements with in-place ActiveX documents.

Previous Home Topic Contents Index Next

IBasicVideo::IsUsingDefaultDestination

IBasicVideo Interface

Determines if the renderer is using the default destination rectangle.

HRESULT IsUsingDefaultDestination( );

Return Values

Returns an HRESULT value. Returns S_OK if using the default destination; otherwise, returns S_FALSE.

Remarks

The IBasicVideo::SetDefaultDestinationPosition method can be used to tell the filter to use the default settings for the destination rectangle. This method returns whether or not they are in use.

Previous Home Topic Contents Index Next

IBasicVideo::IsUsingDefaultSource

IBasicVideo Interface

Determines if the renderer is using the default source rectangle.

HRESULT IsUsingDefaultSource( );

Return Values

Returns an HRESULT value. Returns S_OK if using the default source; otherwise, returns S_FALSE.

Remarks

You can use the IBasicVideo::SetDefaultSourcePosition method to inform the filter to use the default settings for the source rectangle. This method returns whether or not the settings are in use.

Previous Home Topic Contents Index Next

IBasicVideo::put_DestinationHeight

IBasicVideo Interface

Sets the height of the destination rectangle.

HRESULT put_DestinationHeight(
long *DestinationHeight*
);

Parameters

DestinationHeight
[in] New height.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::put_DestinationLeft

IBasicVideo Interface

Sets the destination x-axis origin coordinate.

HRESULT put_DestinationLeft(

long *DestinationLeft*
);

Parameters

DestinationLeft
[in] New x-axis origin.

Return Values

Returns an HRESULT value.

Remarks

Setting this coordinate does not affect the destination rectangle width.

Previous Home Topic Contents Index Next

IBasicVideo::put_DestinationTop

IBasicVideo Interface

Sets the destination y-axis origin coordinate.

HRESULT put_DestinationTop(
long *DestinationTop*
);

Parameters

DestinationTop
[in] New y-axis origin.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::put_DestinationWidth

IBasicVideo Interface

Sets the width of the destination rectangle.

HRESULT put_DestinationWidth(
long *DestinationWidth*
);

Parameters

DestinationWidth
[in] New width.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::put_SourceHeight

IBasicVideo Interface

Sets the height of the source rectangle.

HRESULT put_SourceHeight(
long *SourceHeight*
);

Parameters

SourceHeight
[in] New rectangle height.

Return Values

Returns an HRESULT value.

Remarks

Setting the height of the source rectangle has no effect on the source y-axis coordinate.

Previous Home Topic Contents Index Next

IBasicVideo::put_SourceLeft

IBasicVideo Interface

Sets the source rectangle x-axis coordinate.

HRESULT put_SourceLeft(
long *SourceLeft*
);

Parameters

SourceLeft
[in] The x-axis origin of the source rectangle.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::put_SourceTop

IBasicVideo Interface

Sets the source rectangle y-axis origin coordinate.

HRESULT put_SourceTop(
long *SourceTop*
);

Parameters

SourceTop
[in] New top coordinate.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::put_SourceWidth

IBasicVideo Interface

Sets the source rectangle width.

```
HRESULT put_SourceWidth(
  long SourceWidth
  );
```

Parameters

SourceWidth
[in] New width of the source rectangle.

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IBasicVideo::SetDefaultDestinationPosition

IBasicVideo Interface

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

HRESULT SetDefaultDestinationPosition( );

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::SetDefaultSourcePosition

IBasicVideo Interface

Informs the renderer to use the default source rectangle.

HRESULT SetDefaultSourcePosition( );

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IBasicVideo::SetDestinationPosition

IBasicVideo Interface

Sets the destination rectangle in window coordinates.

HRESULT SetDestinationPosition(
  long *Left*,

long *Top*,
long *Width*,
long *Height*
);

Parameters

Left
[in] The x-axis origin of the destination window.
Top
[in] The y-axis origin of the destination window.
Width
[in] Width of the destination window.
Height
[in] Height of the destination window.

Return Values

Returns an HRESULT value.

Remarks

This method has the same effect as individually calling the IBasicVideo::put_DestinationLeft, IBasicVideo::put_DestinationTop, IBasicVideo::put_DestinationWidth, and IBasicVideo::put_DestinationHeight methods.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IBasicVideo::SetSourcePosition

IBasicVideo Interface

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

HRESULT SetSourcePosition(
long *Left*,
long *Top*,
long *Width*,
long *Height*
);

Parameters

Left

[in] The x-axis origin of the source window.

Top

[in] The y-axis origin of the source window.

Width

[in] Width of the source window.

Height

[in] Height of the source window.

Return Values

Returns an HRESULT value.

Remarks

This method has the same effect as individually calling the IBasicVideo::put_SourceLeft, IBasicVideo::put_SourceTop, IBasicVideo::put_SourceWidth, and IBasicVideo::put_SourceHeight methods.

Setting this coordinate does not affect the destination rectangle height.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder Interface

The **ICaptureGraphBuilder** interface enables you to build capture graphs, preview graphs, file recompression graphs, or even unusual custom graphs easily. See Recompress an AVI File for more information.

When to Implement

Do not implement this interface. DirectShow already does so.

When to Use

Use this interface in your capture and recompression applications to make it easier to build filter graphs.

Methods in Vtable Order

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

ICaptureGraphBuilder methods	Description
SetFiltergraph	Tells the graph builder object which filter graph to use.
GetFiltergraph	Retrieves the filter graph that the builder is using.
SetOutputFileName	Creates the rendering section of the filter graph, which will save bits to disk with the specified file name.
FindInterface	Looks for the specified interface on the filter, upstream and downstream from the filter, and, optionally, on the output pin of the given category.
RenderStream	Connects a source filter's pin, of an optionally specified category, to the rendering filter, and optionally through another filter.
ControlStream	Sends stream control messages to the pin of the specified category on one or more capture filters in a graph.
AllocCapFile	Preallocates a capture file to a specified size.
CopyCaptureFile	Copies the valid media data from the preallocated capture file.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder::AllocCapFile

ICaptureGraphBuilder Interface

Preallocates a capture file to a specified size.

```
HRESULT AllocCapFile(
  LPCOLESTR lpwstr,
  DWORDLONG dwlSize
  );
```

Parameters

lpwstr
[in] Pointer to a Unicode string containing the name of the file to create or resize.
dwSize
[in] Size, in bytes, of the file to be allocated.

Return Values

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

Remarks

The call will fail if the file is read-only. For best capture results, always capture to a defragmented, preallocated capture file that is larger than the size of the capture data.

Note that under the current DirectShow implementation of this interface, a call to this method will be much quicker on a Windows 95 platform than it will be on a Windows NT platform.

ICaptureGraphBuilder::ControlStream

ICaptureGraphBuilder Interface

Sends stream control messages to the pin of the specified category on one or more capture filters in a graph.

```
HRESULT ControlStream(
  const GUID *pCategory,
  IBaseFilter *pFilter,
  REFERENCE_TIME *pstart,
  REFERENCE_TIME *pstop,
  WORD wStartCookie,
  WORD wStopCookie
  );
```

Parameters

pCategory
[in] Pointer to a GUID specifying the output pin category. Typical values include the following. See AMPROPERTY_PIN_CATEGORY for a list of all pin categories. NULL indicates control all capture filters in the graph.
- PIN_CATEGORY_CAPTURE
- PIN_CATEGORY_PREVIEW
- NULL

pFilter
[in] Pointer to an IBaseFilter interface on the filter to control. Specifying NULL controls all capture filters in the graph. You will get one notification for each capture filter.

pstart
[in] Pointer to the start time for capture. NULL means start now. **MAX_TIME** means cancel previous request, or take no action if there is no previous request.

pstop
[in] Pointer to the stop time for capture. NULL means stop now. *MAX_TIME* means cancel previous request, or take no action if there is no previous request.

wStartCookie
[in] Specifies a particular value to be sent when the start occurs.

wStopCookie
[in] Specifies a particular value to be sent when the stop occurs.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns S_FALSE if the stop notification is sent before the last sample sent by the capture filter is rendered, otherwise returns S_OK.

If this method returns S_FALSE, the application might want to wait before destroying the filter graph to allow all samples to pass through the graph and be rendered. Otherwise, samples might be lost.

If a capture filter does not support the IAMStreamControl interface on its preview pin, the filter should return the failure code E_NOINTERFACE in response to a call to this method.

Remarks

Use this method for frame-accurate capture, or for individual control of capture and preview. For example, you could turn off writing of the captured image to disk if you only want to preview the captured image.

This method calls the IAMStreamControl interface on the pins.

This method sends one notification for each filter found with a pin of the specified category.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder::CopyCaptureFile

ICaptureGraphBuilder Interface

Copies the valid media data from the preallocated capture file.

```
HRESULT CopyCaptureFile(
  [in] LPOLESTR lpwstrOld,
  [in] LPOLESTR lpwstrNew,
  [in] int fAllowEscAbort,
  [in] IAMCopyCaptureFileProgress *pCallback
  );
```

Parameters

lpwstrOld
[in] Pointer to a Unicode string containing the source file name.

lpwstrNew
[in] Pointer to a Unicode string containing the destination file name. Valid data is copied to this file.
fAllowEscAbort
[in] TRUE indicates that pressing the ESC key aborts the copy operation, FALSE indicates that this method will ignore that keystroke.
pCallback
[in] Optional pointer to an IAMCopyCaptureFileProgress interface that you implement to show the progress (percentage complete) of the copy operation.

Return Values

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

Remarks

The new file will contain only valid data and therefore can be much shorter than the source file. Typically, you will always capture to the same huge preallocated file and use this method to copy the data you want to save from each capture to a new file.

If you specify *pCallback*, the Progress method on the IAMCopyCaptureFileProgress interface will be called periodically with an integer between 0 and 100 representing the percentage complete.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder::FindInterface

ICaptureGraphBuilder Interface

Looks for the specified interface on the filter, upstream and downstream from the filter, and, optionally, on the output pin of the given category.

```
HRESULT FindInterface(
  const GUID *pCategory,
  IBaseFilter *pf,
  REFIID riid,
  void **ppint
  );
```

Parameters

pCategory
[in] Pointer to a GUID specifying the output pin category. Typical values include the

following. See AMPROPERTY_PIN_CATEGORY for a list of all pin categories. NULL indicates search the output pin regardless of category.

- PIN_CATEGORY_CAPTURE
- PIN_CATEGORY_PREVIEW
- NULL

pf

[in] Pointer to an IBaseFilter interface that is the capture filter.

riid

[in] Reference ID of the desired interface.

ppint

[out] Address of a void pointer. If the interface was found, this method initializes *ppint* so that it contains the address of a pointer to the found interface. Call the Release method to decrement the reference count when you're done with the interface.

Return Values

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

Remarks

This method also looks upstream and downstream of the pin for the interface, to find interfaces on renderers, multiplexers, TV tuners, and so forth.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder::GetFiltergraph

ICaptureGraphBuilder Interface

Retrieves the filter graph that the builder is using.

```
HRESULT GetFiltergraph(
  IGraphBuilder **ppfg
  );
```

Parameters

ppfg

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

Return Values

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

Remarks

This method increments the reference count on the IGraphBuilder interface; be sure to decrement the reference count on **IGraphBuilder** by calling the Release method.

ICaptureGraphBuilder::RenderStream

ICaptureGraphBuilder Interface

Connects a source filter's pin, of an optionally specified category, to the rendering filter, and optionally through another filter.

```
HRESULT RenderStream(
  const GUID *pCategory,
  IUnknown *pSource,
  IBaseFilter *pfCompressor,
  IBaseFilter *pfRenderer
  );
```

Parameters

pCategory
[in] Pointer to a GUID specifying which output pin of the source filter to connect. Typical values include the following. See AMPROPERTY_PIN_CATEGORY for a list of all pin categories. NULL indicates render the only output pin, regardless of category.
- PIN_CATEGORY_CAPTURE
- PIN_CATEGORY_PREVIEW
- NULL

pSource
[in] Pointer to an IBaseFilter or an IPin interface representing either the source filter or an output pin. Source filters are typically a file source filter, such as an AVI file source filter or a capture filter.

pfCompressor
[in] Pointer to an IBaseFilter interface representing the optional compression filter.

pfRenderer
[in] Pointer to an IBaseFilter interface representing the renderer. You can use the *ppf* (multiplexer) parameter from ICaptureGraphBuilder::SetOutputFileName to supply this value.

Return Values

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

Remarks

If you specify PIN_CATEGORY_CAPTURE for *pCategory* and a capture filter for *pSource*, this method instantiates and connects additional required upstream filters, such as TV tuners and crossbars. It then renders the capture pin of *pSource*.

If *pSource* is a pin, then specify NULL for *pCategory* and this method renders the stream from that pin.

If the source filter has only one output pin, specify NULL for *pCategory*.

All required filters must be present in the graph before the application calls this method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICaptureGraphBuilder::SetFiltergraph

ICaptureGraphBuilder Interface

Tells the graph builder object which filter graph to use.

```
HRESULT SetFiltergraph(
  IGraphBuilder *pfg
  );
```

Parameters

pfg

[in] Pointer to an IGraphBuilder interface that specifies the filter graph to use for subsequent calls to the IFilterGraph::AddFilter method.

Return Values

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

Remarks

The graph builder will automatically create a filter graph if you don't call this method. If you call this method after the graph builder has created its own filter graph, the call will fail.

ICaptureGraphBuilder::SetOutputFileName

ICaptureGraphBuilder Interface

Creates the rendering section of the filter graph, which will save bits to disk with the specified file name.

HRESULT SetOutputFileName(
const GUID **pType*,
LPCOLESTR *lpwstrFile*,
IBaseFilter ***ppf*,
IFileSinkFilter ***pSink*
);

Parameters

pType
[in] Pointer to a GUID representing the media subtype. Must be

```
&MEDIASUBTYPE_Avi
```

because AVI is currently the only supported output format.

lpwstrFile
[in] Pointer to a Unicode string containing the output file name.

ppf
[out] Address of a pointer to an IBaseFilter interface representing the multiplexer filter. This method increments the reference count on the **IBaseFilter** interface so you must decrement the reference count by using the Release method on this parameter when done using the filter.

pSink
[out] Address of a pointer to an IFileSinkFilter interface representing the file writer. This method increments the reference count on the **IFileSinkFilter** interface so you must decrement the reference count using Release when done using the filter.

Return Values

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Invalid argument. Audio-video interleaved (AVI) is the only supported output format.
E_OUTOFMEMORY	Out of memory.
E_POINTER	Null pointer argument.
E_UNEXPECTED	Unexpected error occurred.

NOERROR	Success.
S_OK	Instance of the AVI multiplexer filter was successfully created.

Remarks

This method inserts the multiplexer and the file writer into the filter graph and calls IFileSinkFilter::SetFileName to set the output file name.

You can use the *ppf* parameter returned by this method as the *pfRenderer* parameter in calls to RenderStream.

You can use the *pSink* parameter from this method in a call to SetFileName to change the file name set by **ICaptureGraphBuilder::SetOutputFileName**.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IConfigAviMux Interface

The **IConfigAviMux** interface controls how the AVI multiplexer filter writes files out to disk. Microsoft® DirectShow™ currently exposes this interface through the property page of the AVI multiplexer filter and you can use it to set the master stream and compatibility indexes.

IConfigAviMux provides backward compatibility with older Video for Windows® audio-video interleaved (AVI) index formats (idx1) as well as extended AVI 2.0 index formats (indx) to allow for file sizes greater than 1 gigabyte (GB). Set and retrieve the compatibility indexes by using the IConfigAviMux::SetOutputCompatibilityIndex and IConfigAviMux::GetOutputCompatibilityIndex methods. See AVI 2.0 File Format Extensions for more information on AVI 2.0.

When to Implement

DirectShow implements this interface and makes its functionality available to anyone through the property page of an AVI multiplexer filter. However, you can implement this interface yourself if you're writing a filter that takes audio and video streams from a capture graph and generates an AVI file.

When to Use

Use this interface when your application must control how a file is captured.

Methods in Vtable Order

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

IConfigAviMux methods	Description
SetMasterStream	Sets the master stream that other streams must synchronize to after the file is saved to disk.
GetMasterStream	Retrieves the master stream that other streams must synchronize to after the file is written out.
SetOutputCompatibilityIndex	Sets the AVI index format for the file that the multiplexer saves to.
GetOutputCompatibilityIndex	Retrieves the AVI index format for the file that the multiplexer will write to.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IConfigAviMux::GetMasterStream

IConfigAviMux Interface

Retrieves the master stream that other streams must synchronize to after the file is written out.

HRESULT GetMasterStream(
LONG *pStream*
);

Parameters

pStream
[out] Pointer to the master stream.

Return Values

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

Remarks

To eliminate capture drifts that can occur between audio sampling rates and video frame rates, it is recommended that you use combined audio/video cards for capture of large files.

See Also

IConfigAviMux::SetMasterStream

IConfigAviMux::GetOutputCompatibilityIndex

IConfigAviMux Interface

Retrieves the AVI index format for the file that the multiplexer will write to.

HRESULT GetOutputCompatibilityIndex(
BOOL *pfOldIndex*
);

Parameters

pfOldIndex
[out] Value indicating the index format. Returns TRUE if AVI 1.0 (idx1) index format or AVI 2.0 (indx) index format is set; returns FALSE if only AVI 2.0 (indx) index format is set.

Return Values

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

See Also

SetOutputCompatibilityIndex

IConfigAviMux::SetMasterStream

IConfigAviMux Interface

Sets the master stream that other streams must synchronize to after the file is saved to disk.

HRESULT SetMasterStream(

LONG *iStream*
);

Parameters

iStream
[in] Master input stream. Set to –1 to disable.

Return Values

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

Remarks

Different streams in the capture scenario can capture at different rates if you are using separate audio and video capture cards. By specifying a master stream that all others must synchronize to, this method adjusts the frame rate or audio sampling rate to account for drifts in the rates.

To eliminate capture drifts that can occur between audio sampling rates and video frame rates, it is recommended that you use combined audio/video cards for capture of large files.

See Also

IConfigAviMux::GetMasterStream

IConfigAviMux::SetOutputCompatibilityIndex

IConfigAviMux Interface

Sets the AVI index format for the file that the multiplexer saves to.

HRESULT SetOutputCompatibilityIndex(
BOOL *fOldIndex*
);

Parameters

fOldIndex
[in] TRUE indicates either AVI 1.0 (idx1) index format or AVI 2.0 (indx) index format; FALSE indicates only AVI 2.0 (indx) index format.

Return Values

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

Remarks

In addition to backward compatibility with Video for Windows® AVI index formats (idx1), DirectShow also supports extended AVI 2.0 index format (indx), which this method can specify. AVI 2.0 index format allows for increased AVI file size (greater than 1 GB), hierarchical indexing, incremental growth of files, and minimal disk seeks. See AVI 2.0 File Format Extensions for more information on AVI 2.0.

See Also

IConfigAviMux::GetOutputCompatibilityIndex

IConfigInterleaving Interface

The **IConfigInterleaving** interface controls how files are written out to disk and sets interleaving configuration information such as frequency and preroll parameters.

This interface uses the InterleavingMode enumerated data type, which specifies how audio samples and video frames will be saved on a disk. Capture interleaving settings can range from INTERLEAVE_NONE to INTERLEAVE_FULL, depending on whether you will author immediately or later.

DirectShow currently implements this interface on the AVI multiplexer filter; however, you can implement it to set interleaving preferences on other file formats.

When to Implement

DirectShow implements this interface and makes its functionality available to anyone through the property page of an AVI multiplexer filter. However, you can implement this interface yourself when you want to be able to write out other file formats.

When to Use

Video-authoring applications that handle capturing should use this interface when they need to control how audio samples and video frames will be saved on a disk. Applications also use this interface when they need to set interleaving configuration information such as frequency and preroll parameters. (see Amcap.exe for a sample implementation of 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.

IConfigInterleaving methods	Description
put_Mode	Sets how audio samples and video frames will be saved on disk by specifying quality of interleaving.
get_Mode	Retrieves the interleaving quality setting.
put_Interleaving	Sets the audio preroll time and the frequency of the interleaving.
get_Interleaving	Retrieves the audio preroll time and the frequency of the interleaving.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IConfigInterleaving::get_Interleaving

IConfigInterleaving Interface

Retrieves the audio preroll time and the frequency of the interleaving.

HRESULT get_ Interleaving(
REFERENCE_TIME **prtInterleave*,
REFERENCE_TIME **prtPreroll* **);**

Parameters

prtInterleave
[out] Frequency of the streams in the file, in 100-nanosecond units.
prtPreroll
[out] Audio preroll, in 100-nanosecond units.

Return Values

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

See Also

IConfigInterleaving::put_Interleaving

Previous Home Topic Contents Index Next

IConfigInterleaving::get_Mode

IConfigInterleaving Interface

Retrieves the interleaving quality setting.

HRESULT get_ Mode(
InterleavingMode **pMode* **);**

Parameters

mode
[out] Interleaving quality setting specified in the InterleavingMode enumerated data type.

Return Values

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

Remarks

The interleaving mode is specified in the InterleavingMode enumerated data type and is set in the IConfigInterleaving::put_Mode method.

See Also

IConfigInterleaving::put_Mode

Previous Home Topic Contents Index Next

IConfigInterleaving::put_Interleaving

IConfigInterleaving Interface

Sets the audio preroll time and the frequency of the interleaving.

HRESULT put_ Interleaving(
const REFERENCE_TIME **prtInterleave*,

const REFERENCE_TIME **prtPreroll* **);**

Parameters

prtInterleave
[in] Frequency of the streams in the file, in 100-nanosecond units.
prtPreroll
[in] Audio preroll, in 100-nanosecond units.

Return Values

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

Remarks

An audio preroll of 750 milliseconds is recommended when authoring a file for distribution.

The default value for *prtInterleave* is 1000 milliseconds, however you can adjust this. The smaller the number, the larger the file overhead.

See Also

IConfigInterleaving::get_Interleaving

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IConfigInterleaving::put_Mode

IConfigInterleaving Interface

Sets how audio samples and video frames are to be written to disk, by specifying quality of interleaving.

HRESULT put_ Mode(
InterleavingMode *mode* **);**

Parameters

mode
[in] Interleaving quality setting specified in the InterleavingMode enumerated data type.

Return Values

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

Remarks

Applications that require full interleaving authoring quality should specify INTERLEAVE_FULL, INTERLEAVE_CAPTURE, and INTERLEAVE_NONE settings when you will author at a later time.

See Also

IConfigInterleaving::get_Mode

ICreateDevEnum Interface

The **ICreateDevEnum** interface creates an enumerator for a list of objects; typically, these objects are instances of hardware devices. Use this enumerator to access the objects' interfaces, methods, and data. This enumerator supports the same methods as all enumerator interfaces: **Next**, **Skip**, **Reset**, and **Clone**. For more information on using enumerators, see the Win32 IEnumXXXX interface documentation.

The **ICreateDevEnum** interface implements plug-and-play functionality to enumerate devices; DirectShow provides additional plug-and-play functionality through the Win32 IPropertyBag and IPersistPropertyBag interfaces; see the appropriate documentation for more details.

When to Implement

Implement this interface when you want to obtain lists of a specific category of objects. The object must have an internal structure that allows enumeration.

When to Use

Applications need a simple method of finding and creating instances of objects that represent the machine's devices. To obtain these instances, you should first create a system device enumerator object, and then use it to create an enumerator for the particular class of device that interests you. DirectShow typically uses this enumerator to create a list of the machine's video capture devices, which you can then use in your applications or present in some fashion to the user.

Methods in Vtable Order

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

ICreateDevEnum methods	Description
CreateClassEnumerator	Creates a class enumerator for the specified category.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICreateDevEnum::CreateClassEnumerator

ICreateDevEnum Interface

Creates an enumerator for the specified type of object.

```
HRESULT CreateClassEnumerator(
  REFCLSID clsidDeviceClass,
  IEnumMoniker ** ppEnumMoniker,
  DWORD dwFlags
  );
```

Parameters

clsidDeviceClass
[in] REFCLSID value that specifies the type of object or device to enumerate.

ppEnumMoniker
[out] Address of a pointer to the Win32 IEnumMoniker (or derived) interface implemented by the new class enumerator.

dwFlags
[in] Combination of flags that modifies how DirectShow creates the enumerator; for example, an application can use a flag to skip devices that have already been enumerated or that do not support persistant storage of their device properties (CLSID, FriendlyName, and DevicePath).

Return Values

Return one of the following values.

Value	Meaning
NOERROR	Success.
E_OUTOFMEMORY	There is not enough memory available to create a class enumerator.
S_FALSE	The category specified by *clsidDeviceClass* does not exist.

Remarks

The new class enumerator is a nonaggregate object that will run in the same process as the class that called this method.

Previous Home Topic Contents Index Next

ICutListGraphBuilder Interface

The **ICutListGraphBuilder** interface enables you to easily implement one or more cutlist filter graphs. The simplest way to use this object is to add a cutlist by using the ICutListGraphBuilder::AddCutList method, and then use the ICutListGraphBuilder::Render method to create and connect the graph.

You can also use this interface to save your movie to a file. Use the ICutListGraphBuilder::SetOutputFileName method to specify the media subtype (such as MEDIASUBTYPE_Avi) and the file name. **ICutListGraphBuilder** attempts to connect the filters needed to write to this file, so that when you call ICutListGraphBuilder::Render, the filter graph manager creates a filter graph that writes to a file.

See About Cutlists and Using Cutlists for more information.

When to Implement

Do not implement this interface. DirectShow implements it for you.

When to Use

Use this interface in your application when you want to provide cutlist functionality to the user.

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.

ICutListGraphBuilder methods	Description
SetFilterGraph	Sets the filter graph that the cutlist graph builder will use
GetFilterGraph	Retrieves the filter graph the cutlist graph builder is using to implement playback of the cutlist.
AddCutList	Adds a cutlist to the filter graph.
RemoveCutList	Removes a cutlist from a filter graph.
SetOutputFileName	Specifies the output file and media subtype to use when writing the movie to a file.
Render	Creates the final cutlist filter graph.
GetElementFlags	Retrieves the flags of a given element.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::AddCutList

ICutListGraphBuilder Interface

Adds a cutlist to the filter graph.

HRESULT AddCutList(
IStandardCutList **pCutList*,
IPin ***ppPin*
);

Parameters

pCutList
[in] Pointer to the cutlist's IStandardCutList interface that the graph builder might query for more detailed information about the cutlist.

ppPin
[out] Address of a pointer to the output pin of the cutlist filter that this method added to the filter graph. Use this pin as a starting point to build custom cutlist filter graphs. Be sure to decrement the pin's reference count by calling the Release method on the pin before you call the ICutListGraphBuilder::RemoveCutList method. Specify NULL for *ppPin* if you don't need the pin.

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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

When you call the ICutListGraphBuilder::Render method, the ICutListGraphBuilder interface attempts to create a graph that plays back all the cutlists you have added to the graph. You must follow calls to **AddCutList** with a call to **ICutListGraphBuilder::Render** to render the new cutlist in the graph, or your cutlist will not be used. You can wait and call Render once after adding all of your cutlists to the graph.

You can't call IStandardCutList::AddElement on this cutlist after you have given the cutlist to the graph builder by calling this method. The AddElement call will be ignored. Make sure you have called **AddElement** as many times as you need to before calling **ICutListGraphBuilder::AddCutList**.

If you call **AddCutList** to add the cutlist to the filter graph, you must later call ICutListGraphBuilder::RemoveCutList to remove the cutlist from the filter graph. You must do so before releasing the filter graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::GetElementFlags

ICutListGraphBuilder Interface

Retrieves the flags of a given element.

```
HRESULT GetElementFlags(
  IAMCutListElement *pElement,
  LPDWORD lpdwFlags
  );
```

Parameters

pElement
[in] Pointer to the element's IAMCutListElement interface.
lpdwFlags
[out] Retrieved element flags. The flags are a logical combination of members of the

CL_ELEM_FLAGS enumerated data 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	Argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::GetFilterGraph

ICutListGraphBuilder Interface

Retrieves the filter graph that the cutlist graph builder is using to implement playback of the cutlist.

HRESULT GetFilterGraph(
IGraphBuilder ***ppFilterGraph*
);

Parameters

ppFilterGraph
[out] Address of a pointer to the filter graph that the cutlist graph builder is using.

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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

This method is useful, for instance, to acquire the playback graph's IMediaControl and IMediaEvent interfaces.

This method increments the reference count on the filter graph interface pointed to by *ppFilterGraph*. Be sure to decrement the reference count on the filter graph object by calling its Release method when you're done.

See Also

IGraphBuilder

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::RemoveCutList

ICutListGraphBuilder Interface

Removes a cutlist from a filter graph.

HRESULT RemoveCutList(
IStandardCutList **pCutList*
);

Parameters

pCutList
[in] Pointer to the cutlist's IStandardCutList interface that the graph builder might query for more detailed information about the cutlist.

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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

If you called the ICutListGraphBuilder::AddCutList method to add the cutlist to the filter graph, you must call this method before releasing the filter graph.

You should call this method only when the graph is stopped, otherwise resources will not be properly freed.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::Render

ICutListGraphBuilder Interface

Creates the final cutlist filter graph.

HRESULT Render(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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

Call this method to create the final filter graph containing all the previously specified cutlists and file writer filters.

If the graph contains a multiplexer (MUX) filter before you call this method, then this method connects all cutlist source filters to the MUX, and connects the MUX to a single renderer. Otherwise, each cutlist source filter renders through its own renderer. For example, rendering two separate video cutlists without a MUX would give you two video renderer windows for playback.

See Also

ICutListGraphBuilder::AddCutList

Previous Home Topic Contents Index Next

ICutListGraphBuilder::SetFilterGraph

ICutListGraphBuilder Interface

Sets the filter graph that the cutlist graph builder will use.

HRESULT SetFilterGraph(
IGraphBuilder *pFilterGraph*
);

Parameters

pFilterGraph
[in] Pointer to the desired filter graph.

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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

The filter graph can contain filters that the cutlist graph builder will try to use in constructing the graph before adding other filters.

If you've already called this method to give a cutlist to the cutlist graph builder, you can't call this method again.

See Also

IGraphBuilder

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

ICutListGraphBuilder::SetOutputFileName

ICutListGraphBuilder Interface

Specifies the output file and media subtype to use when writing the movie to a file.

HRESULT SetOutputFileName(
const GUID **pType*,
LPCOLESTR *lpwstrFile*,
IBaseFilter ***ppf*,
IFileSinkFilter ***pSink*
);

Parameters

pType
[in] Pointer to a GUID representing the media subtype. Must be &MEDIASUBTYPE_Avi because AVI is currently the only supported output format.

lpwstrFile
[in] Pointer to a Unicode string containing the output file name.

ppf
[in] Address of a pointer to an IBaseFilter interface representing the multiplexer filter. This method increments the reference count on the **IBaseFilter** interface, so you must decrement the reference count by using the Release method on this parameter when you're done using the filter.

pSink
[in] Address of a pointer to an IFileSinkFilter interface representing the file writer. This method increments the reference count on the **IFileSinkFilter** interface, so you must decrement the reference count by using Release when you're done using the filter.

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_OUTOFMEMORY	Could not allocate required memory.
S_OK	Success.

Remarks

This method specifies to the cutlist graph builder that the output of this filter graph should be a file of media subtype *pType*. If you do not call this method, then the cutlist graph builder will attempt to render a graph to play the cutlist to the screen and through the system speakers.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDeferredCommand Interface

The deferred command mechanism allows filters themselves to handle deferred commands. Where they do not, the filter graph manager queues the command until the requested time and then calls the method on the filter (this would result in coarse rather than accurate synchronization). Note that a filter that does handle deferred commands must make them apply to data appearing at that time. Thus a contrast filter asked to change the contrast at time *x* must ensure that it applies the change when processing data time-stamped to be rendered at time *x*; these samples will be processed by the filter somewhat before time *x*.

The IQueueCommand interface provides two methods, InvokeAtStreamTime, which queues commands at stream time, and InvokeAtPresentationTime, which queues commands at presentation time. Both return an **IDeferredCommand** interface to the queued command, by which the application can cancel the command, set a new presentation time for it, or get back an estimate of the likelihood of the filter graph manager being able to run the command on time (implementation of this last method is likely to be highly simplistic in the first release of Microsoft® DirectShow™).

Both the queue and the application will hold a reference count on the object (represented to the application by the **IDeferredCommand** interface), and the object will not be destroyed until both are released. Similarly, calling Release on the **IDeferredCommand** interface is not sufficient to cancel the command because the queue also holds a reference count.

When to Implement

This method is implemented by the filter graph manager to allow deferred processing of commands. It is implemented through a plug-in distributor to pass deferred commands from the application to the filters (through the plug-in distributor for the command that has been queued). You can implement it within your filter if your filter supports queued commands; in this case, applications will need to obtain an IQueueCommand interface directly from your filter. You can use the CDeferredCommand class to help implement this interface.

When to Use

Applications can use this interface to cancel, postpone, get return values from, or determine a confidence level for commands that have been queued for deferred execution by using the

IQueueCommand interface.

Methods in Vtable Order

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

IDeferredCommand methods	Description
Cancel	Cancels a previously queued IQueueCommand::InvokeAtStreamTime or IQueueCommand::InvokeAtPresentationTime request.
Confidence	Returns a confidence value that describes the probability of the deferred command being run on time.
Postpone	Specifies a new invocation time for a previously queued command.
GetHResult	Retrieves the return value from the invoked command.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDeferredCommand::Cancel

IDeferredCommand Interface

Cancels a previously queued IQueueCommand::InvokeAtStreamTime or IQueueCommand::InvokeAtPresentationTime request.

HRESULT Cancel( );

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IDeferredCommand::Confidence

IDeferredCommand Interface

Returns a confidence value on a scale of 0 to 100 that describes the probability that the deferred command will be run on time.

HRESULT Confidence(
LONG **pConfidence*
);

Parameters

pConfidence
Confidence level.

Return Values

Returns an HRESULT value.

Remarks

Higher confidence values indicate a greater probability of timely execution. For example, a command queued at a presentation time that has already passed will return a value of 0. A value of 100 would indicate, with absolute certainty, that the command can be run on time. Commands that are not directly supported by the filter but are queued by the filter graph manager will return a lower confidence value than commands queued directly by the filters that run them. Commands that are queued with times too close to other commands are likely to return lower confidence values. The initial implementation of this method relies mostly on the reporting of the individual filter commands rather than an attempt to estimate resource availability on a filter graph-wide basis.

Previous Home Topic Contents Index Next

IDeferredCommand::GetHResult

IDeferredCommand Interface

Retrieves the return value from the invoked command.

HRESULT GetHResult(
HRESULT* *phrResult*
);

Parameters

phrResult
Retrieved HRESULT value.

IDeferredCommand::Postpone

IDeferredCommand Interface

Specifies a new presentation time for a previously queued command.

HRESULT Postpone(
REFTIME *newtime*
);

Parameters

newtime
New presentation time.

Return Values

Return value is S_OK if completed. Also, if completed, the *phrResult* parameter is set to the result of the deferred command.

IDirectDrawVideo Interface

This interface is on the video renderer and provides information about Microsoft® DirectDraw® with respect to its use by the renderer. For example, the interface allows an application to get details of the surface and any available hardware capabilities. It also allows the application to adjust the surfaces that the renderer should use, and even to set the DirectDraw instance. There is some duplication in this interface with the IDirectDraw interface; however, this interface allows simple access to that information without calling the DirectDraw provider directly.

DirectDraw is not loaded by the renderer until it is connected, and likewise DirectDraw is unloaded only when the renderer is disconnected. When the renderer has allocated the DirectDraw surfaces it will use for video playback, an application can obtain a DDSURFACEDESC structure describing it. By passing in a pointer to a **DDSURFACEDESC** structure, the renderer will fill in the structure with the details of the current surface. If

DirectDraw has not been loaded, the renderer will return E_FAIL. If the renderer is using DCI (the predecessor to DirectDraw), the **DDSURFACEDESC** structure is not filled in but the call will return S_FALSE. The only type of DCI surfaces the renderer uses are primary surfaces.

When to Implement

This interface is implemented by the Microsoft® DirectShow™ video renderer to provide information about DirectDraw surfaces and hardware capabilities.

When to Use

Applications can use this interface to get details of the surface and any available hardware capabilities, adjust the surfaces that the renderer should use, and set the IDirectDraw instance. Applications are allowed to set the **IDirectDraw** instance because DirectDraw can be opened only once per process; this helps resolve conflicts.

Methods in Vtable Order

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

IDirectDrawVideo methods	Description
GetCaps	Retrieves a DirectDraw-defined DDCAPS structure containing the hardware capabilities.
GetEmulatedCaps	Retrieves a DirectDraw-defined DDCAPS structure containing the emulated capabilities.
CanUseOverlayStretch	Determines whether the renderer will check overlay restrictions.
CanUseScanLine	Determines whether the renderer will check the current scan line when drawing.
GetDirectDraw	Retrieves the IDirectDraw interface.
GetFourCCCodes	Retrieves the multimedia format type FOURCC DWORD.
GetSurfaceDesc	Retrieves a description of the DirectDraw surface in use.
GetSurfaceType	Retrieves the actual surface type.
GetSwitches	Retrieves the surface types that the renderer is allowed to use.
SetDefault	Makes the current property settings the global default.
SetDirectDraw	Passes the IDirectDraw interface to a loaded driver.
SetSwitches	Sets the surface types that the renderer is allowed to use.
UseOverlayStretch	Determines whether the renderer should check overlay stretch limitations.
UseScanLine	Determines whether the renderer should check the current scan line when drawing a video.
UseWhenFullScreen	Determines whether DirectShow might change display mode when going to full-screen mode.
WillUseFullScreen	Determines whether DirectShow will change display mode when going to full-screen mode.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawVideo::CanUseOverlayStretch

IDirectDrawVideo Interface

Determines whether the renderer will check overlay restrictions.

HRESULT CanUseOverlayStretch(
long * *UseOverlayStretch*
);

Parameters

UseOverlayStretch
Set to OATRUE if the renderer can use overlay stretching; otherwise, set to OAFALSE.

Return Values

Returns an HRESULT value.

Remarks

For a description of overlay stretching, see IDirectDrawVideo::UseOverlayStretch.

Previous Home Topic Contents Index Next

IDirectDrawVideo::CanUseScanLine

IDirectDrawVideo Interface

Determines whether the renderer will check the current scan line when drawing.

HRESULT CanUseScanLine(
long * *UseScanLine*
);

Parameters

UseScanLine
Set to OATRUE if the renderer can use scan line information; otherwise, set to OAFALSE.

Return Values

Returns an HRESULT value.

Remarks

For a description of the use of scan line detection in the DirectShow video renderer, see IDirectDrawVideo::UseScanLine.

Previous Home Topic Contents Index Next

IDirectDrawVideo::GetCaps

IDirectDrawVideo Interface

Retrieves a DirectDraw-defined DDCAPS structure containing the hardware capabilities.

```
HRESULT GetCaps(
  DDCAPS *pCaps
  );
```

Parameters

pCaps
DDCAPS structure containing the hardware capabilities.

Return Values

Returns an HRESULT value.

Remarks

If the renderer has not loaded DirectDraw, this method returns E_FAIL. DirectDraw is not loaded by the renderer until it is connected, and likewise is unloaded only when the renderer is disconnected.

Previous Home Topic Contents Index Next

IDirectDrawVideo::GetDirectDraw

IDirectDrawVideo Interface

Retrieves the IDirectDraw interface.

HRESULT GetDirectDraw(
LPDIRECTDRAW **ppDirectDraw*
);

Parameters

ppDirectDraw
IDirectDraw interface.

Return Values

Returns an HRESULT value.

Remarks

If an application wants to load DirectDraw but allow the renderer to also allocate surfaces, it can let the renderer load DirectDraw and then obtain a reference-incremented interface to it through this method. The interface returned should be released by the application when it is finished with it.

Previous Home Topic Contents Index Next

IDirectDrawVideo::GetEmulatedCaps

IDirectDrawVideo Interface

Retrieves a DirectDraw-defined DDCAPS structure containing the emulated capabilities.

HRESULT GetEmulatedCaps(
DDCAPS **pCaps*
);

Parameters

pCaps

DDCAPS structure containing the emulated capabilities.

Return Values

Returns an HRESULT value.

Remarks

If the renderer has not loaded DirectDraw, this method returns E_FAIL.

Previous Home Topic Contents Index Next

IDirectDrawVideo::GetFourCCCodes

IDirectDrawVideo Interface

Retrieves the multimedia format type.

```
HRESULT GetFourCCCodes(
  DWORD *pCount,
  DWORD *pCodes
  );
```

Parameters

pCount
Number of FOURCC codes in the *pCodes* array.
pCodes
Array of DWORD media tags formerly used for Microsoft multimedia types.

Return Values

Returns an HRESULT value.

Remarks

In the original Microsoft® Windows® multimedia APIs, media types were tagged with 32-bit values created from four 8-bit characters and were known as FOURCC codes. Because **FOURCC** codes are unique, a one-to-one mapping has been made possible by allocating a range of 4 billion GUIDs representing **FOURCC**s.

This method retrieves the FOURCC codes that the current display driver can support. The number available is obtained by calling the method with a valid *pCount* pointer, but with *pCodes* set to NULL. In this case, the *pCount* variable will be filled in with the number of **FOURCC** codes available. An application can then allocate enough DWORD values for this many **FOURCC** codes and call the method again with the array pointer in *pCodes*.

IDirectDrawVideo::GetSurfaceDesc

IDirectDrawVideo Interface

Retrieves a DDSURFACEDESC structure describing the current DirectDraw surface.

HRESULT GetSurfaceDesc(
DDSURFACEDESC **pSurfaceDesc*
);

Parameters

pSurfaceDesc
DDSURFACEDESC structure describing the current DirectDraw surface.

Return Values

Returns an HRESULT value.

Remarks

If no surface has been allocated, this method will return E_FAIL. If a DCI primary surface is in use, the DDSURFACEDESC structure will not be filled in and the call will return S_FALSE. Surfaces are allocated only when the renderer is paused. Once the renderer has been paused, it cannot release the surfaces when stopped.

IDirectDrawVideo::GetSurfaceType

IDirectDrawVideo Interface

Retrieves the actual surface type as a DirectShow DirectDraw Surface (AMDDS) definition.

HRESULT GetSurfaceType(
DWORD **pSurfaceType*

);

Parameters

pSurfaceType

Field filled in with one bit setting selected from the following list of AMDDS definitions.

AMDDS_NONE	No use for DCI/DirectDraw.
AMDDS_DCIPS	Use DCI primary surface.
AMDDS_PS	Use DirectDraw primary surface.
AMDDS_RGBOVR	RGB overlay surfaces.
AMDDS_YUVOVR	YUV overlay surfaces.
AMDDS_RGBOFF	RGB off-screen surfaces.
AMDDS_YUVOFF	YUV off-screen surfaces.
AMDDS_RGBFLP	RGB flipping surfaces.
AMDDS_YUVFLP	YUV flipping surfaces.
AMDDS_ALL	All the previous flags.
AMDDS_DEFAULT AMDDS_ALL	Use all available surfaces.
AMDDS_YUV	(AMDDS_YUVOFF | AMDDS_YUVOVR | AMDDS_YUVFLP).
AMDDS_RGB	(AMDDS_RGBOFF | AMDDS_RGBOVR | AMDDS_RGBFLP).
AMDDS_PRIMARY	(AMDDS_DCIPS | AMDDS_PS).

Return Values

Returns an HRESULT value.

Remarks

It is not always easy to discover which kind of surface is being used by looking at a DDSURFACEDESC structure. Therefore, an application can call **GetSurfaceType** to retrieve the surface type. The field will be filled in with one bit setting selected from the preceding list of AMDDS definitions.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawVideo::GetSwitches

IDirectDrawVideo Interface

Retrieves the surface types that the renderer is allowed to use.

HRESULT GetSwitches(

DWORD **pSwitches*
);

Parameters

pSwitches
Bit mask containing one or more of the following DirectShow DirectDraw Surface (AMDDS) surface types.

AMDDS_NONE	No use for DCI/DirectDraw.
AMDDS_DCIPS	Use DCI primary surface.
AMDDS_PS	Use DirectDraw primary surface.
AMDDS_RGBOVR	RGB overlay surfaces.
AMDDS_YUVOVR	YUV overlay surfaces.
AMDDS_RGBOFF	RGB off-screen surfaces.
AMDDS_YUVOFF	YUV off-screen surfaces.
AMDDS_RGBFLP	RGB flipping surfaces.
AMDDS_YUVFLP	YUV flipping surfaces.
AMDDS_ALL	All the previous flags.
AMDDS_DEFAULT AMDDS_ALL	Use all available surfaces.
AMDDS_YUV	(AMDDS_YUVOFF | AMDDS_YUVOVR | AMDDS_YUVFLP).
AMDDS_RGB	(AMDDS_RGBOFF | AMDDS_RGBOVR | AMDDS_RGBFLP).
AMDDS_PRIMARY	(AMDDS_DCIPS | AMDDS_PS).

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDirectDrawVideo::SetDefault

IDirectDrawVideo Interface

Makes the current property settings the global default.

HRESULT SetDefault( );

Return Values

Returns an HRESULT value.

Remarks

All properties set through IDirectDrawVideo are specific to that particular instance. Call this method to make properties set on this instance of **IDirectDrawVideo** the global default of all DirectShow instances of this interface. Once called, the current property settings will persist between the subsequent starting of other DirectShow filter graphs and between any computer restarts.

Previous Home Topic Contents Index Next

IDirectDrawVideo::SetDirectDraw

IDirectDrawVideo Interface

Passes the IDirectDraw interface to a loaded driver.

HRESULT SetDirectDraw(
LPDIRECTDRAW *pDirectDraw*
);

Parameters

pDirectDraw
IDirectDraw interface to be passed.

Return Values

Returns an HRESULT value.

Remarks

To have the renderer release a DirectDraw interface previously passed in through **SetDirectDraw**, an application can call **SetDirectDraw**, passing in NULL. However, the renderer will continue using that DirectDraw interface until it is disconnected. Therefore, calling **SetDirectDraw** with a null parameter does not make the renderer stop using it immediately.

This method was created because only one instance of IDirectDraw could be loaded per process in DirectDraw 1.0. If an application wanted to load **IDirectDraw** but allow the renderer to also allocate surfaces, the application could open **IDirectDraw** itself and then pass the interface to the loaded driver through **IDirectDrawVideo::SetDirectDraw**. Alternatively, the application could let the renderer load DirectDraw and then obtain a reference-incremented interface to it through IDirectDrawVideo::GetDirectDraw. Because DirectShow ships with the most recently shipped version of DirectDraw, however, this method is not required unless the application wants to change display modes itself and pass in a DirectDraw object, which the renderer can

then use to allocate surfaces.

IDirectDrawVideo::SetSwitches

IDirectDrawVideo Interface

Sets the surface types that the renderer is allowed to use.

HRESULT SetSwitches(
DWORD *pSwitches*
);

Parameters

pSwitches

Bit mask containing one or more of the following DirectShow DirectDraw Surface (AMDDS) surface types.

AMDDS_NONE	No use for DCI/DirectDraw.
AMDDS_DCIPS	Use DCI primary surface.
AMDDS_PS	Use DirectDraw primary surface.
AMDDS_RGBOVR	RGB overlay surfaces.
AMDDS_YUVOVR	YUV overlay surfaces.
AMDDS_RGBOFF	RGB off-screen surfaces.
AMDDS_YUVOFF	YUV off-screen surfaces.
AMDDS_RGBFLP	RGB flipping surfaces.
AMDDS_YUVFLP	YUV flipping surfaces.
AMDDS_ALL	All the previous flags.
AMDDS_DEFAULT AMDDS_ALL	Use all available surfaces.
AMDDS_YUV	(AMDDS_YUVOFF | AMDDS_YUVOVR | AMDDS_YUVFLP).
AMDDS_RGB	(AMDDS_RGBOFF | AMDDS_RGBOVR | AMDDS_RGBFLP).
AMDDS_PRIMARY	(AMDDS_DCIPS | AMDDS_PS).

Return Values

Returns an HRESULT value.

IDirectDrawVideo::UseOverlayStretch

IDirectDrawVideo Interface

Determines whether the renderer should check overlay stretch limitations.

HRESULT UseOverlayStretch(
long *UseOverlayStretch*
);

Parameters

UseOverlayStretch
Set to OATRUE for the renderer to use overlay stretching; otherwise, set to OAFALSE.

Return Values

Returns an HRESULT value.

Remarks

Some display cards provide the use of overlay surfaces through DirectDraw. An overlay surface is a block of video memory whose contents are overlaid onto the display during the monitor's vertical refresh. DirectShow uses all available overlay surfaces where possible because they typically offer higher-quality video and very fast performance. On some display cards set to relatively high bit depths, the overlay must be displayed on the screen larger than its real size (to accommodate certain display hardware bandwidth limitations). If the overlay is not displayed large enough, certain undesirable effects can be seen on the display (sometimes described as a "fleeting shimmering" effect).

If *UseOverlayStretch* is set to On (the default), DirectShow will ensure the overlay is adequately stretched before displaying it. If it is set to Off, DirectShow will not check that the overlay is adequately stretched, and the user is likely to experience artifacts on the screen (although it will also guarantee that the overlay will be used if possible).

Previous Home Topic Contents Index Next

IDirectDrawVideo::UseScanLine

IDirectDrawVideo Interface

Determines whether the renderer should check the current scan line when drawing a video.

HRESULT UseScanLine(
long *UseScanLine*
);

Parameters

UseScanLine
Long integer value that specifies whether or not to use the scan line information. Set to OATRUE to use scan line information or OAFALSE to ignore it.

Return Values

Returns E_INVALIDARG if the argument is invalid or NOERROR otherwise.

Remarks

If you blit a video image to the screen while the monitor's scan line is scanning over a visible portion of the screen, the complete image will be a composite of the old and new images. This composite is known as a torn video image. You can avoid torn images by waiting until the previous image is complete before blitting the new image. Some video cards can retrieve the scan line's current position; if this information is available, you can have DirectShow try to reduce tearing by waiting until the scan line is offscreen before blitting the new image. Note that checking the scan line location increases load on the processor and can reduce the amount of video frames delivered to the screen. If scan line information is available, DirectShow uses it by default. Set *UseScanLine* to OAFALSE if you want to save processing time at the expense of minor image degradation.

Previous Home Topic Contents Index Next

IDirectDrawVideo::UseWhenFullScreen

IDirectDrawVideo Interface

Determines whether DirectShow might change display mode when going to full-screen mode.

HRESULT UseWhenFullScreen(
long *UseWhenFullScreen*
);

Parameters

UseWhenFullScreen
Set to OATRUE to cause the renderer to use DirectX in full-screen mode; otherwise, set

to OAFALSE.

Return Values

Returns an HRESULT value.

Remarks

When asked to go to full-screen mode, DirectShow has a number of choices. The first choice is to determine if any filters in the graph can support full-screen playback directly; if one can, it will be asked to do so.

The second choice is to automatically add a special full-screen renderer to the filter graph that uses DirectDraw mode-changing services to play back the video. By changing display modes, the video effectively fills more (but not necessarily all) of the display. So, for example, if the current mode is 1024 x 768 pixels, a video might look relatively small, but when displayed in a 320 x 240 display mode it might look very presentable.

The third and final choice is to simply take any renderer supporting the IVideoWindow interface and have its window stretched full-screen. This will typically offer lower performance than the second option (swapping in a full-screen DirectDraw-enabled renderer). If the *UseWhenFullScreen* parameter is set to On (OATRUE), the window will always be stretched full-screen for full-screen playback; if set to Off (the default), the filter graph manager is free to swap in the DirectDraw-enabled full-screen renderer.

Previous Home Topic Contents Index Next

IDirectDrawVideo::WillUseFullScreen

IDirectDrawVideo Interface

Determines whether DirectShow will change display mode when going to full-screen mode.

HRESULT WillUseFullScreen(
long * *UseWhenFullScreen*
);

Parameters

UseWhenFullScreen
Set to OATRUE if DirectShow is set to use DirectX in full-screen mode; otherwise, set to OAFALSE.

Return Values

Returns an HRESULT value.

Remarks

For a description of this feature, see IDirectDrawVideo::UseWhenFullScreen.

IDistributorNotify Interface

The **IDistributorNotify** interface is an optional interface for use by plug-in distributors to notify them of changes in the filter graph state.

When to Implement

Implement this interface when writing a plug-in distributor (PID) that is aggregated with the filter graph manager if you want the PID to receive notifications of control and changes in the composition of filter graphs.

PIDs may often use methods on the filter graph manager. During a call to an IDistributorNotify method, do not take any lock that may be held by another code path that calls the filter graph manager. To do so could result in a deadlock.

When to Use

The filter graph manager queries for this interface on any plug-in distributors that it aggregates. If found, it calls the appropriate Run, Pause, or SetSyncSource method before calling them on the IBaseFilter interface of each filter. It calls the NotifyGraphChange method whenever a filter is added or removed, or connections are changed.

Methods in Vtable Order

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

IDistributorNotify methods	Description
Stop	Called when the filter graph is entering a stopped state.
Pause	Called when the filter graph is entering a paused state.
Run	Called when the filter graph is entering a running state.
SetSyncSource	Called when a new clock is registered.
NotifyGraphChange	Called when the set of filters in the filter graph change or their connections change.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDistributorNotify::NotifyGraphChange

IDistributorNotify Interface

Called when the set of filters in the filter graph change or their connections change.

HRESULT NotifyGraphChange(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 is called whenever an IFilterGraph::AddFilter, IFilterGraph::RemoveFilter, or IFilterGraph::ConnectDirect method is called or a method is called that will lead to one of these being called (such as IGraphBuilder::RenderFile).

Make sure you use Release on any held filters that have been removed at this point. For performance reasons, PIDs might choose not to rescan the filters until the PIDs actually need the interfaces, because there might be several separate notifications sent. However, it is important to release any cached interfaces immediately.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDistributorNotify::Pause

IDistributorNotify Interface

Called when the filter graph is entering a paused state.

HRESULT Pause(void);

Return Values

Returns S_OK if the transition is complete; otherwise, returns one of the following values.

Value	Meaning
Error value	Transition failed.
S_FALSE	Transition is not complete, but no error has occurred.

Remarks

This method is called before the filters are notified.

Previous Home Topic Contents Index Next

IDistributorNotify::Run

IDistributorNotify Interface

Called when the filter graph is entering a running state.

HRESULT Run(
REFERENCE_TIME *tStart*
);

Parameters

tStart
Stream-time offset that will be passed to every filter's IBaseFilter::Run parameter.

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 is called before the filters are notified.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDistributorNotify::SetSyncSource

IDistributorNotify Interface

Called when a new clock is registered.

HRESULT SetSyncSource(
IReferenceClock * *pClock*
);

Parameters

pClock
[in] Pointer to the IReferenceClock interface.

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 is called before the filters are notified. Make sure to use AddRef on the *pClock* parameter if the plug-in distributor intends to hold it beyond this method call.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDistributorNotify::Stop

IDistributorNotify Interface

Called when the filter graph is entering a stopped state.

HRESULT Stop(void);

Return Values

Returns S_OK if the transition is complete; otherwise, returns one of the following values.

Value	Meaning
Error value	Transition failed.
S_FALSE	Transition is not complete, but no error has occurred.

Remarks

This method is called before the filters are notified.

Previous Home Topic Contents Index Next

IDvdControl Interface

The **IDvdControl** interface controls the playback and search mechanisms of a digital versatile disc that contains one or more video movies.

The DVD file system is very different than a format such as CD-ROM, which contains a linear series of tracks not easily customizable by the author. The author of DVD-formatted media files can control track layout and navigation much more precisely; the media file itself comprises many parts and control mechanisms, which enables the author to arrange and rearrange the playback order as needed. You can locate a specific portion of a file by time (in hours, minutes,

seconds, and frames) or by chapter value.

A media file is made up of a list of program chains (PGCs), each of which are made up of a list of programs, each of which is made up of a list of cells, each of which is made up of a list of video object units (VOBUs), each of which is made up of a list of packs, and each of which is made up of actual MPEG data.

To obtain a copy of the **DVD-Video specification** , "DVD Specifications for Read-Only Disc, Part 3, Video Specifications," contact Toshiba Corporation at 1-1, Shibaura 1-Chrome, Minato-ku, Tokyo 105-01, Japan, Tel. +81-3-5444-9580, Fax. +81-3-5444-9430.

When to Implement

Do not implement this interface. DirectShow implements it for you.

When to Use

Use this interface to control playback (set root drive, play, stop, and so forth) or use DVD-specific functionality such as menus and buttons when playing DVD-Video files.

Methods in Vtable Order

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

IDvdControl methods	Description
TitlePlay	Finds the media file with the specified title index and plays it back.
ChapterPlay	Plays the media file with the specified title index and chapter value.
TimePlay	Plays the media file with the specified title index, starting at the specified time.
StopForResume	Transitions playback to the DVD_DOMAIN_Stop state after saving resume information.
GoUp	Halts playback of the current media file and starts playback of the designated previous program group chain (PGC).
TimeSearch	Halts playback of the current chapter and starts playback from the specified time in the same media file.
ChapterSearch	Halts playback of the current chapter and starts playback from the specified chapter value in the same media file.
PrevPGSearch	Halts playback of the current chapter and starts playback from the previous chapter in the same PGC.
TopPGSearch	Halts playback of the current chapter and starts playback from the first chapter within the title.
NextPGSearch	Halts playback of the current chapter and starts playback from the next chapter within the title.

ForwardScan	Search forward through the current disc at the specified speed.
BackwardScan	Search backward through the current disc at the specified speed.
MenuCall	Displays the specified menu on the screen.
Resume	Continues playback of a media file, if the file is paused.
UpperButtonSelect	Selects the upper directional button from the displayed menu.
LowerButtonSelect	Selects the lower directional button from the displayed menu.
LeftButtonSelect	Selects the left directional button from the displayed menu.
RightButtonSelect	Selects the right directional button from the displayed menu.
ButtonActivate	Activates the selected button.
ButtonSelectAndActivate	Selects and activates the specified button.
StillOff	Resumes updating the display, which stops still-store mode.
PauseOn	Pauses the current media file playback.
PauseOff	Unpauses the current media file playback.
MenuLanguageSelect	Sets the displayed language for navigation menus.
AudioStreamChange	Sets the current audio stream.
SubpictureStreamChange	Enables or disables picture-in-picture mode and sets the subpicture to the specified source.
AngleChange	Sets the new display angle.
ParentalLevelSelect	Sets the access level for the current media file.
ParentalCountrySelect	Sets the current country for controlling access levels.
KaraokeAudioPresentationModeChange	Sets the audio playback mode to karaoke.
VideoModePreferrence	Sets the video display mode the user prefers.
SetRoot	Sets the root directory containing the DVD-Video volume.
MouseActivate	Selects and activates a DVD button in response to a mouse click.
MouseSelect	Selects a DVD button in response to mouse movement.
ChapterPlayAutoStop	Start playing at the specified chapter within the specified title and play the number of chapters specified.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::AngleChange

IDvdControl Interface

Sets the new display angle.

HRESULT AngleChange(
ULONG *ulAngle*
);

Parameters

ulAngle
[in] Value of the new angle, which must be between 1 and 9.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::AudioStreamChange

IDvdControl Interface

Sets the current audio stream.

HRESULT AudioStreamChange(
ULONG *nAudio*
);

Parameters

nAudio

[in] Value that specifies the track to use, which must be between 0 and 7.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::BackwardScan

IDvdControl Interface

Search backward through the current disc at the specified speed.

```
HRESULT BackwardScan(
  double dwSpeed
  );
```

Parameters

dwSpeed
[in] Value that specifies how quickly DirectShow will search through the media file. This value is a multiplier, where 1.0 is the authored speed, so a value of 2.5 would search backward at two and one-half times the authored speed.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ButtonActivate

IDvdControl Interface

Activates the selected button.

HRESULT ButtonActivate(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

"Selecting" a DVD button simply highlights the button but does not "activate" the button. Selecting is the Windows equivalent of tabbing to a button but not pressing the space bar or enter key. Activating is the Windows equivalent of pressing the space bar or enter key after tabbing to a button.

See Also

IDvdControl::ButtonSelectAndActivate

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ButtonSelectAndActivate

IDvdControl Interface

Selects and activates the specified button.

HRESULT ButtonSelectAndActivate(
ULONG *uiButton*
);

Parameters

uiButton
[in] Value that specifies the button that will be selected and activated, which must be between 1 and 36.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

Electronic remote control devices typically have a number of buttons that activate various functions of a DVD playback unit. Typically, you call this method when a user clicks a button on the control device; DirectShow then indicates that the button was selected (by playing a sound or changing a graphic, for example) and calls methods appropriate to which button was selected, such as ButtonActivate.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ChapterPlay

IDvdControl Interface

Plays the media file with the specified title index and chapter value.

HRESULT ChapterPlay(
ULONG *uiTitle*,
ULONG *uiChapter*
);

Parameters

uiTitle
[in] Value that specifies the title number DirectShow will play back; this value must be between 1 and 99.
uiChapter
[in] Value that specifies the point within the specified title where DirectShow will start playback; this value must be between 1 and 999.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ChapterPlayAutoStop

IDvdControl Interface

Instructs the DVD player to start playing at the specified chapter within the specified title and play the number of chapters specified.

HRESULT ChapterPlayAutoStop(
ULONG *ulTitle*,
ULONG *ulChapter*,
ULONG *ulChaptersToPlay*
);

Parameters

ulTitle
[in] Title number for playback.
ulChapter
[in] Chapter number to start playback.
ulChaptersToPlay
[in] Number of chapters to play from the start chapter.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

Chapters range from 1 to 999.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ChapterSearch

IDvdControl Interface

Halts playback of the current chapter and starts playback from the specified chapter within the same title.

HRESULT ChapterSearch(
ULONG *Chapter*
);

Parameters

Chapter
Chapter value that specifies the point where playback will begin.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::ForwardScan

IDvdControl Interface

Search forward through the current disc at the specified speed.

HRESULT ForwardScan(
double *dwSpeed*
);

Parameters

dwSpeed
[in] Value that specifies how quickly DirectShow will search through the media file. This

value is a multiplier, where 1.0 is the authored speed, so a value of 2.5 would search forward at two and one-half times the authored speed.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::GoUp

IDvdControl Interface

Halts playback of the current media file and starts playback of the designated previous program group chain (PGC).

HRESULT GoUp(void);

Return Values

Returns an HRESULT value.

Remarks

Each PGC is associated with a previous PGC at authoring time, which this method sets as the new playback file.

Previous Home Topic Contents Index Next

IDvdControl::KaraokeAudioPresentationModeCha

IDvdControl Interface

Sets the audio playback mode to karaoke.

HRESULT KaraokeAudioPresentationModeChange(
ULONG *ulMode*
);

Parameters

ulMode
[in] Requested audio playback mode.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

Karaoke support is currently not implemented.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::LeftButtonSelect

IDvdControl Interface

Selects the left directional button from the displayed menu.

HRESULT LeftButtonSelect(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

On physical or electronic remote control devices, there is often a group of four directional buttons used for certain types of operations (such as menu navigation). This method tells DirectShow that something (the user, probably) triggered the left directional button.

"Selecting" a DVD button simply highlights the button but does not "activate" the button. Selecting is the Windows equivalent of tabbing to a button but not pressing the space bar or enter key. Activating is the Windows equivalent of pressing the space bar or enter key after tabbing to a button.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::LowerButtonSelect

IDvdControl Interface

Selects the lower directional button from the displayed menu.

HRESULT LowerButtonSelect(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

On physical or electronic remote control devices, there is often a group of four directional buttons used for certain types of operations (such as menu navigation). This method tells DirectShow that something (the user, probably) triggered the lower directional button.

"Selecting" a DVD button simply highlights the button but does not "activate" the button. Selecting is the Windows equivalent of tabbing to a button but not pressing the space bar or enter key. Activating is the Windows equivalent of pressing the space bar or enter key after tabbing to a button.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::MenuCall

IDvdControl Interface

Displays the specified menu on the screen.

HRESULT MenuCall(
DVD_MENU_ID *MenuID*
);

Parameters

MenuID
[in] Value that specifies the menu to display. Member of the DVD_MENU_ID enumerated data type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::MenuLanguageSelect

IDvdControl Interface

Sets the displayed language for navigation menus.

HRESULT MenuLanguageSelect(
LCID *LanguageCode*
);

Parameters

LanguageCode
Value that specifies the new language.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

This method selects the default language for menus. Call this method only from the DVD stop state (DVD_DOMAIN_Stop). Applications specify languages with Windows standard LCIDs. LCIDs can be created from ISO-639 codes with:

```
MAKELCID( MAKELANGID(wISO639LangID ,SUBLANG_DEFAULT ), SORT_DEFAULT )
```

You might have to use 'jp' instead of 'ja' for the ISO639 code for Japanese.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::MouseActivate

IDvdControl Interface

Selects and activates a DVD button in response to a mouse click.

HRESULT MouseActivate(
POINT *point* **);**

Parameters

point
[in] Specified point within the display window.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

This method checks the specified point within the display window to see if it is within a current DVD button's highlight rectangle. If it is, this method selects and then activates the button.

DVD buttons do not all necessarily have highlight rectangles. Button rectangles can overlap, and the rectangles do not always correspond to the visual representation of DVD buttons.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::MouseSelect

IDvdControl Interface

Selects a DVD button in response to mouse movement.

HRESULT MouseSelect(
POINT *point* **);**

Parameters

point
[in] Specified point within the display window.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

This method checks the specified point within the display window to see if it is within a current DVD button's highlight rectangle. If it is, this method selects the button.

DVD buttons do not all necessarily have highlight rectangles. Button rectangles can overlap, and the rectangles do not always correspond to the visual representation of DVD buttons.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::NextPGSearch

IDvdControl Interface

Halts playback of the current chapter and starts playback from the next chapter within the

title.

HRESULT NextPGSearch(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

IDvdControl::ParentalCountrySelect

IDvdControl Interface

Sets the current country for controlling access levels.

HRESULT ParentalCountrySelect(
WORD *wCountry*
);

Parameters

wCountry
[in] Value that specifies the current country.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

See Also

ParentalLevelSelect

Previous Home Topic Contents Index Next

IDvdControl::ParentalLevelSelect

IDvdControl Interface

Sets the access level for the current media file.

HRESULT ParentalLevelSelect(
ULONG *ulParentalLevel*
);

Parameters

ulParentalLevel
Value that specifies the current media file access level, which must be **DVD_PARENTAL_LEVEL_1** (child safe, most restriction), **DVD_PARENTAL_LEVEL_6** (theatrical), **DVD_PARENTAL_LEVEL_8** (adult, no restriction), or disabled (0xffffffff).

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

This method sets the current user's access level; this access level determines what media files the user can play back. Higher levels can play lower level content; lower levels can't play higher-level content. For example, adults can watch child-safe content, but children can't watch adult content.

The DVD Navigator filter provides no restriction on setting the parental level. DVD player applications can enforce restrictions on the parental level setting, such as providing password

protection for raising the current parental level. The DVD Navigator is initialized by default to parental level 8 (no restriction).

To disable parental management, pass 0xffffffff for *ulParentalLevel*. If parental management is disabled, then the player will play the first program chain (PGC) in a parental block regardless of parental IDs.

See Also

ParentalCountrySelect

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::PauseOff

IDvdControl Interface

Unpauses the current media file playback, which returns it to normal playback.

HRESULT PauseOff(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

If the media file wasn't paused in playback, this method does nothing.

See Also

IDvdControl::PauseOn

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::PauseOn

IDvdControl Interface

Pauses the current media file playback.

HRESULT PauseOn( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

This method freezes playback and any internal timers, similar to IMediaControl::Pause.

If the media file wasn't running, this method does nothing.

See Also

IDvdControl::PauseOff

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::PrevPGSearch

IDvdControl Interface

Halts playback of the current chapter and starts playback from the previous chapter in the same PGC.

HRESULT PrevPGSearch(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

IDvdControl::Resume

IDvdControl Interface

Continues playback of a media file, if the file is paused.

HRESULT Resume( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

If the file is stopped or already running, this method does nothing.

This method returns to title playback in DVD_DOMAIN_Title. Applications typically call this method after MenuCall which puts the DVD Navigator in DVD_DOMAIN_VideoTitleSetMenu or DVD_DOMAIN_VideoManagerMenu.

Previous Home Topic Contents Index Next

IDvdControl::RightButtonSelect

IDvdControl Interface

Selects the right directional button from the displayed menu.

HRESULT RightButtonSelect( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

On physical or electronic remote control devices, there is often a group of four directional buttons used for certain types of operations (such as menu navigation). This method tells DirectShow that something (the user, probably) triggered the right directional button.

"Selecting" a DVD button simply highlights the button but does not "activate" the button. Selecting is the Windows equivalent of tabbing to a button but not pressing the space bar or enter key. Activating is the Windows equivalent of pressing the space bar or enter key after tabbing to a button.

© 1997 Microsoft Corporation. All rights reserved. Terms of Use.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::SetRoot

IDvdControl Interface

Sets the root directory containing the DVD-Video volume.

HRESULT SetRoot(
LPCWSTR *pszPath* **);**

Parameters

pszPath
[in] Directory name to set as the root directory.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

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

Remarks

The current state must be DVD_DOMAIN_Stop when you call this method.

If you haven't set the root directory before calling IDvdControl::TitlePlay, the first drive starting from C: that contains a VIDEO_TS directory in the top level directory will be used as the root.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::StillOff

IDvdControl Interface

Resumes updating the display, which stops still-store mode.

HRESULT StillOff( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

If the display image wasn't in still-store mode, this method does nothing.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::StopForResume

IDvdControl Interface

Transitions playback to the DVD_DOMAIN_Stop state after saving resume information.

HRESULT StopForResume(void);

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Remarks

If no file is playing or paused, this method does nothing.

The DVD Navigator filter transfers to the stopped state, but the filter graph remains in DirectShow's Run state.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::SubpictureStreamChange

IDvdControl Interface

Enables or disables picture-in-picture mode and sets the subpicture to the specified source.

HRESULT SubpictureStreamChange(
ULONG *nSubPicture*,
BOOL *bDisplay*
);

Parameters

nSubPicture
Value that specifies the source of the subpicture, which must be between 0 and 31, or 63.
bDisplay
Boolean value that specifies whether the subpicture is enabled; TRUE makes the subpicture visible and FALSE hides it.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::TimePlay

IDvdControl Interface

Plays the media file with the specified title index, starting at the specified time.

```
HRESULT TimePlay(
  ULONG uiTitle,
  ULONG bcdTime
  );
```

Parameters

uiTitle
Value that specifies the title number DirectShow will play back; this value must be between 1 and 99.

bcdTime
Pointer to the DVD_TIMECODE structure where DirectShow will start playback.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::TimeSearch

IDvdControl Interface

Halts playback of the current chapter and starts playback from the specified time in the same

media file.

HRESULT TimeSearch(
ULONG *bcdTime*
);

Parameters

bcdTime
Pointer to the DVD_TIMECODE structure where DirectShow will start playback.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::TitlePlay

IDvdControl Interface

Finds the media file with the specified title index and plays it back.

HRESULT TitlePlay(
ULONG *uiTitle*
);

Parameters

uiTitle
Value that specifies the title number DirectShow will play back; this value must be between 1 and 99.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdControl::TopPGSearch

IDvdControl Interface

Halts playback of the current chapter and starts playback from the first chapter within the title.

HRESULT TopPGSearch( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
E_UNEXPECTED	DVD is not initialized.
S_OK	Success.

Previous Home Topic Contents Index Next

IDvdControl::UpperButtonSelect

IDvdControl Interface

Selects the upper directional button from the displayed menu.

HRESULT UpperButtonSelect( );

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
S_OK	Success.

Remarks

On physical or electronic remote control devices, there is often a group of four directional buttons used for certain types of operations (such as menu navigation). This method tells DirectShow that something (the user, probably) triggered the upper directional button.

"Selecting" a DVD button simply highlights the button but does not "activate" the button. Selecting is the Windows equivalent of tabbing to a button but not pressing the space bar or enter key. Activating is the Windows equivalent of pressing the space bar or enter key after tabbing to a button.

Previous Home Topic Contents Index Next

IDvdControl::VideoModePreferrence

IDvdControl Interface

Sets the video display mode the user prefers.

HRESULT VideoModePreferrence(
ULONG *ulPreferredDisplayMode*
);

Parameters

ulPreferredDisplayMode
[in] Value that specifies the new display mode for DVD content. Member of the DVD_PREFERRED_DISPLAY_MODE enumerated data type.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. Typical return values might include one of the following:

Value	Meaning
E_FAIL	Failure.
E_INVALIDARG	Input argument is invalid.
E_NOTIMPL	Method is not supported.
S_OK	Success.

Remarks

This method changes the default video window's aspect ratio, and may also specify a default aspect ratio conversion mechanism.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdGraphBuilder Interface

The **IDvdGraphBuilder** interface enables you to easily build a filter graph for DVD-Video playback.

When to Implement

Do not implement this interface. DirectShow's DVD-Video graph builder object implements it for you.

When to Use

Use this interface in your application to build a filter graph for DVD-Video playback.

Methods in Vtable Order

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

IDvdGraphBuilder methods	Description
GetFiltergraph	Retrieves the IGraphBuilder interface for the filter graph used by the DVD-Video graph builder object.
GetDvdInterface	Retrieves specific interface pointers in the DVD-Video playback graph to make DVD-Video playback development easier.
RenderDvdVideoVolume	Completes building a filter graph according to user specifications for playing back a DVD-Video volume.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdGraphBuilder::GetDvdInterface

IDvdGraphBuilder Interface

Retrieves specific interface pointers in the DVD-Video playback graph to make DVD-Video playback development easier.

```
HRESULT GetDvdInterface(
  REFIID riid,
  void **ppvIF
  );
```

Parameters

riid
[in] IID of the required interface.

ppvIF
[out] Address of a pointer to the retrieved interface.

Return Values

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

The current DirectShow implementation return values include the following:

Value	Meaning
E_INVALIDARG	The *ppvIF* parameter is invalid.
E_NOINTERFACE	The *riid* parameter value is not supported.
VFW_E_DVD_GRAPHNOTREADY	Graph has not been built through the IDvdGraphBuilder::RenderDvdVideoVolume method.

Remarks

You'll typically use this method to get the IDvdControl interface to control playback of a DVD-Video volume, or to get the IAMLine21Decoder interface to turn closed captioning on and off.

Remember to release the interface by using the following code when you're done with it:

```
*ppvIF->Release();
```


Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdGraphBuilder::GetFiltergraph

IDvdGraphBuilder Interface

Retrieves the IGraphBuilder interface for the filter graph used by the DVD-Video graph builder object.

HRESULT GetFiltergraph(
IGraphBuilder ***ppGB*
);

Parameters

ppGB
[out] Address of a pointer to the IGraphBuilder interface that the DVD-Video graph builder object is using.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns E_INVALIDARG if *ppGB* is invalid.

Remarks

Remember to release the IGraphBuilder interface by using the following code when you're done with it:

```
*ppGB->Release();
```

IDvdGraphBuilder::RenderDvdVideoVolume

IDvdGraphBuilder Interface

Completes building a filter graph according to user specifications for playing back a DVD-Video volume.

HRESULT RenderDvdVideoVolume(
LPCWSTR *lpcwszPathName*,
DWORD *dwFlags*,
AM_DVD_RENDERSTATUS **pStatus*
);

Parameters

lpcwszPathName
[in] Path name for the DVD-Video volume to play. Can be NULL.
dwFlags
[in] Member of the AM_DVD_GRAPH_FLAGS enumerated data type indicating the type of decoder (hardware or software or a mix of hardware and software) to include in the filter graph. Maximum hardware decoding (AM_DVD_HWDEC_PREFER) is the default.
pStatus
[out] Pointer to the retrieved AM_DVD_RENDERSTATUS structure, which returns indication of any failure.

Return Values

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

Value	Meaning
E_INVALIDARG	The *dwFlags* parameter specifies conflicting options or *pStatus* is a bad pointer.
S_FALSE	Graph has been built, but not perfectly. The *pStatus* parameter provides details of the problem(s).
S_OK	Success. Playback graph built successfully, all streams rendered, and the DVD-Video volume was specified or found.
VFW_E_DVDDEC_NOTENOUGH	AM_DVD_HWDEC_ONLY or AM_DVD_SWDEC_ONLY was specified and there weren't enough hardware or software decoders to decode all streams.

VFW_E_DVDGRAPH_RENDERFAIL	Some basic error occurred in building the graph. Possibilities include the DVD Navigator filter or the video or audio renderer not instantiating, a trivial connection or pin enumeration failing, or none of the streams rendering.

Remarks

The AM_DVD_RENDERSTATUS structure reflects failure codes for this method. Reasons for this method returning S_FALSE include the following:

- The graph has been completely built, but one of the following is true.
  - VPE mixing doesn't work (application didn't use the AM_DVD_NOVPE flag). In this case, the application will have enough information to tell the user that the video won't be visible unless a TV is connected to the NTSC out port of the DVD-Video decoder (presumably hardware decoder in this case).
  - Video decoder doesn't produce Line21 data. The application can put up a warning or informative message that closed captioning is not available because of the decoder.
  - No volume path specified and DVD Navigator didn't locate any DVD-Video volume to be played. Application can ask the user to insert a DVD-Video disc, if none is available in the drive when playback is started.
- Some streams didn't render or didn't render completely. Volume can be partially played back. The application can indicate to the user that some of the streams can't be played.

This method builds the graph without any knowledge of the DVD-Video file or volume to play back. The DVD-Video graph builder builds the graph even if *lpwszPathName* is NULL or if the DVD Navigator filter does not find a default DVD-Video volume to play back. An application can later specify the volume by using the IDvdControl::SetRoot method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo Interface

The **IDvdInfo** interface enables an application to query for attributes of available digital versatile disc (DVD) titles and the DVD player status. It also allows for control of a DVD player beyond Annex J in the DVD specification.

When to Implement

Do not implement this interface.

When to Use

Use this interface in your application to retrieve details about a DVD-Video or about the current

state of the DVD player filter graph.

Methods in Vtable Order

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

IDvdInfo methods	Description
GetCurrentDomain	Retrieves the current DVD domain of the DVD player.
GetCurrentLocation	Retrieves the current playback location.
GetTotalTitleTime	Retrieves the total playback time for the current title.
GetCurrentButton	Retrieves the number of available buttons and the currently selected button number.
GetCurrentAngle	Retrieves the number of available angles and the currently selected angle number.
GetCurrentAudio	Retrieves the number of available audio streams and the number of the currently selected audio stream.
GetCurrentSubpicture	Retrieves the number of available subpicture streams, the currently selected subpicture stream number, and whether the subpicture display is disabled.
GetCurrentUOPS	Retrieves which IDvdControl methods are valid.
GetAllSPRMs	Retrieves the current contents of all system parameter registers (SPRMs).
GetAllGPRMs	Retrieves the current contents of all general parameter registers (GPRMs).
GetAudioLanguage	Retrieves the language of the specified audio stream within the current title.
GetSubpictureLanguage	Retrieves the language of the specified subpicture stream within the current title.
GetTitleAttributes	Retrieves attributes of all video, audio, and subpicture streams for the specified title, including menus.
GetVMGAttributes	Retrieves attributes of all video, audio, and subpicture streams for video manager (VMG) menus.
GetCurrentVideoAttributes	Retrieves the video attributes for the current title or menu.
GetCurrentAudioAttributes	Retrieves the audio attributes for the stream in the current title or menu.
GetCurrentSubpictureAttributes	Retrieves the video attributes for the stream in the current title or menu.
GetCurrentVolumeInfo	Retrieves the current DVD volume information.
GetDVDTextInfo	Retrieves the TXTDT_MG structure, which can contain text descriptions for title name, volume name, producer name, vocalist name, and so on, in various languages.
GetPlayerParentalLevel	Retrieves the current parental level and country code settings for the DVD player.
GetNumberOfChapters	Retrieves the number of chapters that are defined for a given title.
GetTitleParentalLevels	Retrieves the parental levels that are defined for a particular title.
GetRoot	Retrieves the root directory that is set in the player.

Previous Home Topic Contents Index Next

IDvdInfo::GetAllGPRMs

IDvdInfo Interface

Retrieves the current contents of all general parameter registers (GPRMs).

HRESULT GetAllGPRMs(
GPRMARRAY *pRegisterArray* **);**

Parameters

pRegisterArray
[out] Pointer to the retrieved array of general parameter registers (**GPRMARRAY**).

Return Values

Returns an HRESULT value that depends on the implementation of the interface. See IDvdInfo::GetCurrentDomain for a list of typical return values for the methods exposed by this interface.

Remarks

Use of GPRMs is title specific.

Previous Home Topic Contents Index Next

IDvdInfo::GetAllSPRMs

IDvdInfo Interface

Retrieves the current contents of all system parameter registers (SPRMs).

HRESULT GetAllSPRMs(
SPRMARRAY *pRegisterArray* **);**

Parameters

pRegisterArray
[out] Pointer to the retrieved array of system parameter registers (**SPRMARRAY**).

Return Values

Returns an HRESULT value that depends on the implementation of the interface. See IDvdInfo::GetCurrentDomain for a list of typical return values for the methods exposed by this interface.

Remarks

See the DVD-Video specification for use of individual registers.

Previous Home Topic Contents Index Next

IDvdInfo::GetAudioLanguage

IDvdInfo Interface

Retrieves the language of the specified audio stream within the current title.

HRESULT GetAudioLanguage(
ULONG *nStream*,
LCID **pLanguage* **);**

Parameters

nStream
[in] Stream number.
pLanguage
[out] Pointer to the retrieved language.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. See IDvdInfo::GetCurrentDomain for a list of typical return values for the methods exposed by this interface.

Remarks

This method does not return languages for menus. This method sets the value pointed to by *pLanguage* to zero if the stream does not include language. Call the Win32 **GetLocaleInfo** API as follows to create a human-readable string name from *pLanguage*: