```
GetLocaleInfo(*pLanguage, LOCALE_SENGLANGUAGE, pszString, cbSize);
```


Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentAngle

IDvdInfo Interface

Retrieves the number of available angles and the currently selected angle number.

HRESULT GetCurrentAngle(
ULONG **pnAnglesAvailable*,
ULONG **pnCurrentAngle* **);**

Parameters

pnAnglesAvailable
[out] Pointer to the retrieved number of available angles. If the value pointed to equals 1, then the current video does not contain multiple angles.
pnCurrentAngle
[out] Pointer to the retrieved current angle number.

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

Angles are interleaved video streams that presumably contain the same scene shot with different camera angles.

This method returns an error unless the domain is DVD_DOMAIN_Title.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentAudio

IDvdInfo Interface

Retrieves the number of available audio streams and the number of the currently selected audio stream.

HRESULT GetCurrentAudio(
ULONG **pnStreamsAvailable*,
ULONG **pnCurrentStream* **);**

Parameters

pnStreamsAvailable
[out] Pointer to the retrieved number of available audio streams
pnCurrentStream
[out] Pointer to the current stream number.

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 returns an error unless the domain is DVD_DOMAIN_Title.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentAudioAttributes

IDvdInfo Interface

Retrieves the audio attributes for the stream in the current title or menu.

HRESULT GetCurrentAudioAttributes(
DVD_AudioATR **pATR* **);**

Parameters

pATR
[out] Pointer to the retrieved audio attributes.

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 information about DVD_AudioATR.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentButton

IDvdInfo Interface

Retrieves the number of available buttons and the currently selected button number.

```
HRESULT GetCurrentButton(
  ULONG *pnButtonsAvailable,
  ULONG *pnCurrentButton );
```

Parameters

pnButtonsAvailable
[out] Pointer to the number of buttons available.
pnCurrentButton
[out] Pointer to the number of the current button.

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

If buttons are not present this method returns zero for both *pnButtonsAvailable* and *pnCurrentButton*.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentDomain

IDvdInfo Interface

Retrieves the current DVD domain of the DVD player.

HRESULT GetCurrentDomain(
DVD_DOMAIN *pDomain **);**

Parameters

pDomain
[out] Pointer to the current domain which is a member of the DVD_DOMAIN enumerated 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_OUTOFMEMORY	Out of memory (insufficient buffer space).
E_POINTER	NULL pointer argument.
E_UNEXPECTED	DVD is not initialized or domain is not DVD_DOMAIN_Title.
E_UOP_PROHIBITED	Requested action cannot occur at this point in the movie due to the authoring of the current DVD-Video disc.
S_OK	Success.
VFW_E_DVD_INVALIDDOMAIN	Requested action is not supported on this domain (DVD_DOMAIN).

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentLocation

IDvdInfo Interface

Retrieves the current playback location.

HRESULT GetCurrentLocation(
DVD_PLAYBACK_LOCATION **pLocation* **);**

Parameters

pLocation
[out] Pointer to the retrieved playback location information in a DVD_PLAYBACK_LOCATION structure.

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.

The default implementation of this method returns E_UNEXPECTED if the current domain is not DVD_DOMAIN_Title.

Remarks

This method retrieves information sufficient to restart playback of a video from the current playback location in titles that don't explicitly disable seeking to the current location.

This method returns an error unless the domain is DVD_DOMAIN_Title.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentSubpicture

IDvdInfo Interface

Retrieves the number of available subpicture streams, the currently selected subpicture stream number, and whether the subpicture display is disabled.

HRESULT GetCurrentSubpicture(
ULONG **pnStreamsAvailable*,
ULONG **pnCurrentStream*,
BOOL **pIsDisabled* **);**

Parameters

pnStreamsAvailable
[out] Pointer to the retrieved number of available subpicture streams.

pnCurrentStream
[out] Pointer to the retrieved number of the currently selected subpicture stream.
pIsDisabled
[out] Pointer to a value indicating whether the subpicture display is disabled.

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

Subpicture streams authored as forcedly activated streams will be displayed even if the application has disabled subpicture display with the IDvdControl::SubpictureStreamChange method.

This method returns an error unless the domain is DVD_DOMAIN_Title.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentSubpictureAttributes

IDvdInfo Interface

Retrieves the video attributes for the stream in the current title or menu.

HRESULT GetCurrentSubpictureAttributes(
DVD_SubpictureATR **pATR* **);**

Parameters

pATR
[out] Pointer to the retrieved subpicture attributes.

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 information about DVD_SubpictureATR.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentUOPS

IDvdInfo Interface

Retrieves which IDvdControl methods are currently valid.

HRESULT GetCurrentUOPS(
VALID_UOP_SOMTHING_OR_OTHER **pUOP* **);**

Parameters

pUOP
[out] Pointer to the retrieved valid user operations (UOP).

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 is useful because DVD titles can enable or disable individual user operations at almost any point during playback.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentVideoAttributes

IDvdInfo Interface

Retrieves the video attributes for the current title or menu.

HRESULT GetCurrentVideoAttributes(

DVD_VideoATR **pATR* **);**

Parameters

pATR
[out] Pointer to the retrieved video attributes.

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 information about DVD_VideoATR.

Previous Home Topic Contents Index Next

IDvdInfo::GetCurrentVolumeInfo

IDvdInfo Interface

Retrieves the current DVD volume information.

HRESULT GetCurrentVolumeInfo(
ULONG **pNumOfVol*,
ULONG **pThisVolNum*,
DVD_DISC_SIDE **pSide*,
ULONG **pNumOfTitles* **);**

Parameters

pNumOfVol
[out] Pointer to the retrieved number of volumes in a volume set.
pThisVolNum
[out] Pointer to the retrieved volume number for this root directory.
pSide
[out] Pointer to the retrieved current disc side (DVD_DISC_SIDE).
pNumOfTitles
[out] Pointer to the retrieved number of titles available in this volume.

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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetDVDTextInfo

IDvdInfo Interface

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.

HRESULT GetDVDTextInfo(
  BYTE **pTextManager*,
  ULONG *cbBufSize*,
  ULONG **pcbActualSize* **);**

Parameters

pTextManager
[out, size_is(cbBufSize)] Pointer to the retrieved text manager.
cbBufSize
[in] Size of the buffer for *pTextManager*, in bytes.
pcbActualSize
[out] Pointer to a value containing the number of bytes of data returned.

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

If the supplied buffer size in *cbBufSize* is too small for the data, (for example if *cbBufSize* equals zero), then this method returns E_OUTOFMEMORY and sets the value pointed to by *pcbActualSize* to the required size.

Refer to Section 4.1.6 and Annex A of the DVD-Video specification for more information.

Previous Home Topic Contents Index Next

IDvdInfo::GetNumberOfChapters

IDvdInfo Interface

Retrieves the number of chapters that are defined for a given title.

HRESULT GetNumberOfChapters(
ULONG *ulTitle*,
ULONG **pNumberOfChapters*
);

Parameters

ulTitle
[in] Title for which to retrieve the number of chapters.
pNumberOfChapters
[out] Retrieved number of chapters for the specified title.

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.

IDvdInfo::GetPlayerParentalLevel

IDvdInfo Interface

Retrieves the current parental level and country code settings for the DVD player.

HRESULT GetPlayerParentalLevel(
ULONG **pParentalLevel*,
ULONG **pCountryCode*
);

Parameters

pParentalLevel
[out] Pointer to a value indicating the current parental level.
pCountryCode
[out] Pointer to a value indicating the current country code.

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 Table 3.3.4-1 of the DVD-Video specification for the defined parental levels.

See ISO3166 : Alpha-2 Code for the country codes.

Valid Parental Levels are 1 through 8 if parental management is enabled, 0xffffffff if parental management is disabled.

Previous Home Topic Contents Index Next

IDvdInfo::GetRoot

IDvdInfo Interface

Retrieves the root directory that is set in the player.

```
HRESULT GetRoot(
  LPTSTR pRoot,
  ULONG cbBufSize,
  ULONG *pcbActualSize
  );
```

Parameters

pRoot
[out, size_is(cbBufSize)] Pointer to the buffer to receive the root string.
cbBufSize
[in] Size of buffer passed in, in bytes.
pcbActualSize
[out] Pointer to a value containing the size of the actual data returned.

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

If a valid root was found, this method returns the root string. Otherwise, it returns zero for *pcbActualSize*, indicating that a valid root directory has not been found or initialized.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetSubpictureLanguage

IDvdInfo Interface

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

HRESULT GetSubpictureLanguage(
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*.

```
GetLocaleInfo(*pLanguage, LOCALE_SENGLANGUAGE, pszString, cbSize);
```

This method returns an error unless the domain is DVD_DOMAIN_Title.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IDvdInfo::GetTitleAttributes

IDvdInfo Interface

Retrieves attributes of all video, audio, and subpicture streams for the specified title, including menus.

HRESULT GetTitleAttributes(
  ULONG *nTitle*,
  DVD_ATR **pATR* **);**

Parameters

nTitle
[in] Requested title number. Specify 0xFFFFFFFF for the current title.
pATR
[out] Pointer to the retrieved attributes structure (DVD_ATR).

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.

Previous Home Topic Contents Index Next

IDvdInfo::GetTitleParentalLevels

IDvdInfo Interface

Retrieves the parental levels that are defined for a particular title.

HRESULT GetTitleParentalLevels(
ULONG *ulTitle*,
ULONG **pParentalLevels*
);

Parameters

ulTitle
[in] Title for which parental levels are requested.
pParentalLevels
[out] Logical combination of the parental levels defined for the title. Valid parental levels are DVD_PARENTAL_LEVEL_8, DVD_PARENTAL_LEVEL_6, and DVD_PARENTAL_LEVEL_1.

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.

Previous Home Topic Contents Index Next

IDvdInfo::GetTotalTitleTime

IDvdInfo Interface

Retrieves the total playback time for the current title.

HRESULT GetTotalTitleTime(
ULONG **pTotalTime* **);**

Parameters

pTotalTime
[out] Pointer to the total time in DVD_TIMECODE format, which includes hours, minutes, seconds, and frames.

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 works only for simple linear movies.

IDvdInfo::GetVMGAttributes

IDvdInfo Interface

Retrieves attributes of all video, audio, and subpicture streams for video manager (VMG) menus.

HRESULT GetVMGAttributes(
DVD_ATR *pATR* **);**

Parameters

pATR
[out] Pointer to the retrieved attributes structure (DVD_ATR).

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

The video manager contains a separate group of streams, such as the DVD_MENU_Title menus and are not associated with any particular title number.

IEnumFilters Interface

The IFilterGraph::EnumFilters method returns the enumerator interface. It is based on the COM IEnum style of enumerators.

When to Implement

This interface is implemented on the filter graph manager and is not intended for implementation by developers.

When to Use

This interface is used by applications or other filters to determine what filters exist in the filter graph.

Methods in Vtable Order

IUnknown methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.
IEnumFilters methods	**Description**
Next	Retrieves the specified number of filters in the enumeration sequence.
Skip	Skips a specified number of filters in the enumeration sequence.
Reset	Resets the enumeration sequence to the beginning.
Clone	Retrieves a duplicate enumerator containing the same enumeration state as the current one.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumFilters::Clone

IEnumFilters Interface

Retrieves a duplicate enumerator containing the same enumeration state as the current one.

```
HRESULT Clone(
  IEnumFilters ** ppEnum
  );
```

Parameters

ppEnum
[out] Duplicate of the enumerator.

Return Values

No return value.

Remarks

This method produces two enumerators (the original and the duplicate); both are set at the same filter. After they are created, however, they are independent; therefore, calling the IEnumFilters::Next method for one does not affect the other.

IEnumFilters::Next

IEnumFilters Interface

Retrieves the specified number of filters in the enumeration sequence.

HRESULT Next(
ULONG *cFilters*,
IBaseFilter ** *ppFilter*,
ULONG * *pcFetched*
);

Parameters

cFilters
[in] Number of filters to place.
ppFilter
[out] Array in which to place IBaseFilter pointers.
pcFetched
[out] Actual number of filters placed in the array.

Return Values

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

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

Remarks

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

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumFilters::Reset

IEnumFilters Interface

Resets the enumeration sequence to the beginning.

HRESULT Reset(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 affects the return value of the next call to the IEnumFilters::Next method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumFilters::Skip

IEnumFilters Interface

Skips a specified number of filters in the enumeration sequence.

HRESULT Skip(
ULONG *cFilter*
);

Parameters

cFilter
[in] Number of filters to skip.

Return Values

No return value.

Remarks

This method affects the return value of the next call to the IEnumFilters::Next method.

Previous Home Topic Contents Index Next

IEnumMediaTypes Interface

The **IEnumMediaTypes** interface enumerates the preferred formats for a pin.

When to Implement

This interface must be implemented and made available by the IPin::EnumMediaTypes method. The CBasePin::EnumMediaTypes member function automatically does this in the DirectShow™ class library and uses the CEnumMediaTypes class to create the enumerator object.

When to Use

This interface is normally used by a connecting pin to determine the media type when negotiating a connection. It can also be passed through to other pins, either upstream or downstream of the filter, when intervening filters do not have a list of preferred media types. For example, a transform-inplace filter might pass the IEnumMediaTypes interface of a downstream input pin to the connecting output pin of the upstream filter, instead of providing its own **IEnumMediaTypes** interface.

Methods in Vtable Order

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

IEnumMediaTypes methods	Description
Next	Retrieves the specified number of items in the enumeration sequence.
Skip	Skips a specified number of elements in the enumeration sequence.
Reset	Resets the enumeration sequence to the beginning.
Clone	Returns another enumerator containing the same enumeration state as the current one.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumMediaTypes::Clone

IEnumMediaTypes Interface

Returns another enumerator containing the same enumeration state as the current one.

```
HRESULT Clone(
  IEnumMediaTypes ** ppEnum
  );
```

Parameters

ppEnum
[out] New copy of the enumerator.

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.

Previous Home Topic Contents Index Next

IEnumMediaTypes::Next

IEnumMediaTypes Interface

Retrieves the specified number of items in the enumeration sequence.

```
HRESULT Next(
  ULONG cMediaTypes,
  AM_MEDIA_TYPE ** ppMediaTypes,
  ULONG * pcFetched
  );
```

Parameters

cMediaTypes
[in] Number of media types to place.
ppMediaTypes
[out] Array in which to place the pointers to the next media type.
pcFetched
[out] Actual count passed.

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

To use this method, pass an array of pointers to media types. (If you want only a single media type, you can pass a pointer to a media type pointer in place of an array of media type pointers.) The interface allocates the necessary AM_MEDIA_TYPE structures and initializes them with the variable format block. Free each media type by calling DeleteMediaType, which will free the format block and the media type itself.

Previous Home Topic Contents Index Next

IEnumMediaTypes::Reset

IEnumMediaTypes Interface

Resets the enumerator to the beginning so that the next call to the IEnumMediaTypes::Next method returns, at a minimum, the first media type (if any) in the enumeration.

HRESULT Reset(void);

Return Values

Returns S_OK if successful; otherwise, returns S_FALSE.

Previous Home Topic Contents Index Next

IEnumMediaTypes::Skip

IEnumMediaTypes Interface

Skips a specified number of elements in the enumeration sequence.

HRESULT Skip(
ULONG *cMediaTypes*
);

Parameters

cMediaTypes
[in] Number of media type elements to skip.

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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumPins Interface

The IBaseFilter::EnumPins method returns this interface. It is based on the IEnumX protocol of the Component Object Model (COM).

Note that actions that cause the number of pins to change might cause the enumerator to fail.

When to Implement

This interface must be implemented and made available by the IBaseFilter::EnumPins method. The CBaseFilter::EnumPins member function automatically does this in the DirectShow™ class library and uses the CEnumPins class to create the enumerator object.

When to Use

This interface is normally used by the filter graph manager when connecting filters. It can, however, be used by an application that must find the pins associated with filters in the filter graph—for example, to add another filter to the graph.

Methods in Vtable Order

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

IEnumPins methods	Description
Next	Puts pointers to IPin interfaces for the next pins into the specified array.
Skip	Skips the specified number of pins.
Reset	Resets the position to the beginning so that the next call to the IEnumPins::Next method returns, at a minimum, the first pin of the filter.
Clone	Provides another enumerator, which is a duplicate of the current one.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumPins::Clone

IEnumPins Interface

Makes a copy of the enumerator. This allows the calling application to retain two positions in the list of pins.

HRESULT Clone(
IEnumPins ** *ppEnum*
);

Parameters

ppEnum
[out] New enumerator, which is a copy of this enumerator.

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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumPins::Next

IEnumPins Interface

Places pointers to IPin interfaces into the specified array.

```
HRESULT Next(
  ULONG cPins,
  IPin ** ppPins,
  ULONG * pcFetched
  );
```

Parameters

cPins
[in] Number of pins to place.
ppPins
[out] Array in which to place the interface pointers.
pcFetched
[out] Actual number of pins placed in the array.

Return Values

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

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

Remarks

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

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumPins::Reset

IEnumPins Interface

Resets the enumerator to the beginning so that the next call to the IEnumPins::Next method returns, at a minimum, the first pin of the filter.

HRESULT Reset(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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumPins::Skip

IEnumPins Interface

Skips the specified number of pins.

HRESULT Skip(
ULONG *cPins*
);

Parameters

cPins
[in] Number of pins to skip.

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 affects the next call to the IEnumPins::Next method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumRegFilters Interface

The purpose of the mapper is to help the filter graph manager avoid loading filters when attempting to build a filter graph to render a given media type. By looking at filter properties recorded in the registry, the number of filters that must be loaded and tried can be reduced.

The IFilterMapper::EnumMatchingFilters method returns the enumerator that enumerates the filters that match specific requirements. The enumerator returns descriptors of filters, including the globally unique identifiers (GUIDs) that the Microsoft® Win32® CoCreateInstance function can instantiate. The filters are not loaded. The **IEnumRegFilters** interface is a Component Object Model (COM) enumerator.

When to Implement

This interface is implemented by the filter mapper and need not be implemented elsewhere.

When to Use

Although the filter graph manager is the primary user of this interface, applications can also use it to determine available filters in the system — for example, to construct a unique filter graph by adding and connecting filters itself, or to allow users to choose from a list of available filters.

Methods in Vtable Order

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

IEnumRegFilters methods	Description
Next	Fills an array with the next filters that meet the requirements.
Skip	Skips a specified number of elements in the enumeration sequence.
Reset	Makes the Next method start again, beginning at the first filter.
Clone	Returns another enumerator containing the same enumeration state as the current one.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumRegFilters::Clone

IEnumRegFilters Interface

Creates another enumerator with the same enumeration state as the current one.

```
HRESULT Clone(
  IEnumRegFilters **ppEnum
  );
```

Parameters

ppEnum
[out] Pointer to the duplicate enumerator interface.

Return Values

Returns an HRESULT value. Currently returns E_NOTIMPL.

Remarks

Currently, this method is not implemented.

Previous Home Topic Contents Index Next

IEnumRegFilters::Next

IEnumRegFilters Interface

Fills the array with descriptions of the next set of filters (specified by the *cFilters* parameter) that meet the requirements specified upon creation of the enumerator.

HRESULT Next(
ULONG *cFilters*,
REGFILTER ** *apRegFilter*,
ULONG * *pcFetched*
);

Parameters

cFilters
[in] Number of filters.
apRegFilter
[out] Pointer to an array of REGFILTER pointers.
pcFetched
[out] Actual number of filters passed.

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 calling application must use the Microsoft Win32 CoTaskMemFree function to free each REGFILTER pointer returned in the array. Do not free the **Name** member of the **REGFILTER** structure separately, because **IEnumRegFilters::Next** allocates memory for this string as part of the **REGFILTER** structure.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumRegFilters::Reset

IEnumRegFilters Interface

Resets the enumerator so that the next call to the IEnumRegFilters::Next method begins again at the first filter, if any.

HRESULT Reset(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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IEnumRegFilters::Skip

IEnumRegFilters Interface

Skips a specified number of items in the enumeration sequence.

HRESULT Skip(
ULONG *celt*
);

Parameters

celt
[in] Number of items to skip.

Return Values

Returns an HRESULT value. Currently returns E_NOTIMPL.

Remarks

Currently, this method is not implemented.

Previous Home Topic Contents Index Next

IFileClip Interface

The **IFileClip** interface provides a simple way for an application to create one or more cuts from a single media file, or to create blank cuts. Blank (empty or null) cuts are useful to either stop playback for a specified time, or to make a placeholder for a cut that the cutlist can't play.

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.

IFileClip methods	Description
SetFileAndStream	Initializes the clip object with the specified media file and stream number or makes an empty clip for producing null elements.
CreateCut	Creates a cutlist element.
GetMediaType	Retrieves the clip's media type structure.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileClip::CreateCut

IFileClip Interface

Creates a cutlist element.

HRESULT CreateCut(
IAMCutListElement ***ppElement*,
REFERENCE_TIME *mtTrimIn*,
REFERENCE_TIME *mtTrimOut*,
REFERENCE_TIME *mtOrigin*,
REFERENCE_TIME *mtLength*,
REFERENCE_TIME *mtOffset*
);

Parameters

ppElement
[out] Address of a pointer to the IAMCutListElement interface of the created cutlist element.
mtTrimIn
[in] Trimin (beginning) position for the cut.
mtTrimOut
[in] Trimout (ending) position for the cut.
mtOrigin
[in] Clip origin. Must be zero.
mtLength
[in] Length of clip. Must be *mtTrimOut* minus *mtTrimIn*.
mtOffset
[in] Offset of clip. Must be zero.

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

All of the times specified in this method are relative to the media clip rather than to the cutlist.

To make an empty cut, first create an empty (null) clip by calling the

IFileClip::SetFileAndStream method as illustrated by the following code fragment. Then, create a cut of the desired duration (*n*) to indicate you want to do nothing for *n* units of time.

```
IAMCutListElement *pElement;

SetFileAndStream(NULL, -1);
CreteCut(&pElement, 0, n, 0, n, 0);
```

See Using Cutlists for more information about using cutlists and the cutlist interfaces from an application.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileClip::GetMediaType

IFileClip Interface

Retrieves the clip's media type structure.

HRESULT GetMediaType(
AM_MEDIA_TYPE **pmt*
);

Parameters

pmt
[out] Pointer to the AM_MEDIA_TYPE structure describing the video clip.

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

IFileClip::SetFileAndStream

IFileClip Interface

Initializes the clip object with the specified media file and stream number or makes an empty clip for producing null elements.

HRESULT SetFileAndStream(
LPWSTR *wstrFileName*,
DWORD *streamNum*
);

Parameters

wstrFileName
[in] Name of the file from which to initialize the clip. Must be an AVI or .wav file. Specify NULL to make an empty (null) clip.

streamNum
[in] Stream number (AVI files only) within the specified file from which to initialize the clip. Must be zero. AVI files with more than one stream of any type are not supported; clips must be from the first stream (stream 0). Specify –1 for empty elements.

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 opens the file to verify the format and media type (which you can find by using the IFileClip::GetMediaType method).

Use the following call to make an empty clip.

```
SetFileAndStream(NULL, -1);
```

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSinkFilter Interface

The **IFileSinkFilter** interface is implemented on filters that write media streams to a file. A file sink filter in a video capture filter graph, for instance, writes the output of the video compression filter to a file. Typically, the application running this filter graph will want to allow the user to enter the name of the file to be written to. This interface enables the communication of this information.

IFileSinkFilter2 replaces this interface unless you need to maintain backward compatibility with ActiveMovie 1.0.

When to Implement

If a filter needs the name of an output file, it should expose this interface to allow an application to set the file name. Note that there is currently no base class implementation of this interface.

When to Use

Any application that must set the name of the file into which the file sink filter will write should use this interface to get and set the file name.

Methods in Vtable Order

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

IFileSinkFilter methods	Description
SetFileName	Sets the name of the file into which media samples will be written.
GetCurFile	Retrieves the name of the current file into which media samples will be written (the sink file).

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSinkFilter::GetCurFile

IFileSinkFilter Interface

Retrieves the name of the current file into which media samples will be written (the sink file).

```
HRESULT GetCurFile(
  LPOLESTR *ppszFileName,
  AM_MEDIA_TYPE *pmt
  );
```

Parameters

ppszFileName
[out] Name of the file set to receive media samples.
pmt
Type of media samples to be written to the file.

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

If a file name is not assigned, this method returns E_FAIL.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSinkFilter::SetFileName

IFileSinkFilter Interface

Sets the name of the file into which media samples will be written.

HRESULT SetFileName(
LPCOLESTR *pszFileName*,
const AM_MEDIA_TYPE **pmt*
);

Parameters

pszFileName
[in] Name of the file to receive the media samples.
pmt
[in] Type of media samples to be written to the file, and the media type of the sink filter's input pin.

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

If the *pszFileName* parameter names a nonexistent file, the file will be created. If it names an existing file, the sink filter will overwrite the file without first deleting it.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSinkFilter2 Interface

The **IFileSinkFilter2** interface derives from the IFileSinkFilter interface and replaces it unless you need backward compatibility with ActiveMovie™ 1.0. Like **IFileSinkFilter**, filters that write media streams to a file implement this interface. A file sink filter in a video capture filter graph, for instance, saves the output of the video compression filter to a file. Typically, the application running this filter graph should allow the user to enter the name of the file to which to save the data. This interface enables you to communicate this information. **IFileSinkFilter2** adds the option to determine whether the file it writes should destroy an existing file of the

same name. In the video capture case, do not destroy a file you've already created, because preallocating file space takes valuable time. By default, the new file does not destroy the old one. Otherwise, destroy the original file to make sure the file you author doesn't contain garbage.

When to Implement

A filter should implement this interface when it needs the name of an output file or needs to set options for that file. Implement the older IFileSinkFilter interface if you need to make your filter compatible with ActiveMovie 1.0. Note that there is currently no base class implementation of this interface.

When to Use

When an application must set the name of the file into which the file sink filter will write, it should use this interface to get and set the file name or change options. Use the older IFileSinkFilter interface if you need to make your application compatible with ActiveMovie 1.0.

Methods in Vtable Order

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

IFileSinkFilter methods	Description
SetFileName	Sets the name of the file into which media samples will be written.
GetCurFile	Retrieves the name of the current file into which media samples will be written (the *sink file*).

IFileSinkFilter2 methods	Description
SetMode	Determines whether the file writer destroys the file when it creates the new one.
GetMode	Retrieves whether the file writer destroys the file when it creates the new one.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSinkFilter2::GetMode

IFileSinkFilter2 Interface

Retrieves whether the file writer destroys the file when it creates the new one.

```
HRESULT GetMode(
  DWORD *dwFlags
  );
```

Parameters

dwFlags
[out] Pointer to the retrieved flags. Currently, the only defined flag is AM_FILE_OVERWRITE, which indicates that the file should be destroyed; zero indicates that the file will be left alone.

Return Values

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

Previous Home Topic Contents Index Next

IFileSinkFilter2::SetMode

IFileSinkFilter2 Interface

Determines whether the file writer destroys the file when it creates the new one.

```
HRESULT SetMode(
  DWORD dwFlags
  );
```

Parameters

dwFlags
[in] Currently, the only defined flag is AM_FILE_OVERWRITE, which indicates that the file writer should destroy the file. Specify zero for *dwFlags* to leave the file alone.

Return Values

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

Previous Home Topic Contents Index Next

IFileSourceFilter Interface

The **IFileSourceFilter** interface is exposed by source filters to set the file name and media type of the media file that they are to render. It is an abbreviated version of the COM IPersistFile interface. If the file has a type that can be determined by the algorithm described in Registering a Custom File Type, the recommended file source filter CLSID is used when the filter graph manager attempts to render the filter graph.

When to Implement

If a filter needs the name of a file to open, it should expose this interface to allow an application to set the file name. Note that there is no base class implementation of this interface.

When to Use

An application that inserts file source filters directly must query for this interface and set the file name. Normally, the filter graph manager uses this interface when an application calls IGraphBuilder::RenderFile. The Graphedt.exe tool queries for the IFileSourceFilter interface and prompts for a file name if it finds it.

Methods in Vtable Order

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

IFileSourceFilter methods	Description
Load	Loads the source filter with the file.
GetCurfile	Retrieves the current file.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSourceFilter::GetCurfile

IFileSourceFilter Interface

Collects information about the file to open.

```
HRESULT GetCurfile(
  LPOLESTR *ppszFileName,
  AM_MEDIA_TYPE *pmt
  );
```

Parameters

ppszFileName
[out] Path to the loaded file.
pmt
[out] Media type.

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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFileSourceFilter::Load

IFileSourceFilter Interface

Loads a media file.

```
HRESULT Load(
  LPCOLESTR pszFileName,
  const AM_MEDIA_TYPE *pmt
  );
```

Parameters

pszFileName
[in] Absolute path of the file to open.
pmt
[in] Media type of the file. This can be NULL.

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 is an initialization method for the interface. It is not designed to load multiple files, and any calls to this method after the first call will fail.

You should implement this method to load the file specified by *pszFileName*.

Note that the name in *pszFileName* need not actually be a disk file name (that is, one you could pass to the Microsoft® Win32® CreateFile function, for example). It could also be a URL name. The URL moniker filter uses IFileSourceFilter to retrieve its URL name, and IGraphBuilder::AddSourceFilter (and hence IGraphBuilder::RenderFile) handles this correctly. **IGraphBuilder::AddSourceFilter** returns a specific error (ERROR_FILE_NOT_FOUND) upon not finding the file, which indicates that the file specified does not exist and not that the filter does not exist.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph Interface

The **IFilterGraph** interface is an abstraction representing a graph of filters. All filters in the graph share the same clock. They might or might not also be connected and stream data between them. This interface allows filters to be joined into a graph and operated as a unit. Unlike the IGraphBuilder interface, this interface does not use heuristics to connect and build the filter graph.

When to Implement

This interface is implemented on the filter graph manager and is not intended for implementation by developers.

When to Use

Applications should not use this interface directly but instead should use the IGraphBuilder interface, which inherits this interface.

Methods in Vtable Order

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

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

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::AddFilter

IFilterGraph Interface

Adds a filter to the graph and names it by using the *pName* parameter.

```
HRESULT AddFilter(
  IBaseFilter * pFilter,
  LPCWSTR pName
  );
```

Parameters

pFilter
[in] Filter to add to the graph.

pName
[in] Name of the filter.

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 name of the filter can be NULL, in which case the filter graph manager will generate a name. If the name is not NULL and is not unique, this method will modify the name in an attempt to generate a new unique name. If this is successful, this method returns VFW_S_DUPLICATE_NAME. If it cannot generate a unique name, it returns VFW_E_DUPLICATE_NAME.

AddFilter calls the filter's IBaseFilter::JoinFilterGraph method to inform the filter that it has been added. **AddFilter** must be called before attempting to use the IGraphBuilder::Connect, IFilterGraph::ConnectDirect, or IGraphBuilder::Render method to connect or render pins belonging to the added filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::ConnectDirect

IFilterGraph Interface

Connects the two pins directly (without intervening filters).

```
HRESULT ConnectDirect(
  IPin * ppinOut,
  IPin * ppinIn,
  const AM_MEDIA_TYPE * pmt
  );
```

Parameters

ppinOut
[in] Output pin.
ppinIn
[in] Input pin.
pmt
[in] Media type to use for the connection (optional; that is, can be NULL).

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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::Disconnect

IFilterGraph Interface

Disconnects this pin.

```
HRESULT Disconnect(
  IPin * ppin
  );
```

Parameters

ppin
[in] Pin to disconnect.

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 does not completely break the connection. To completely break the connection, both ends must be disconnected.

This method results in a successful no operation (no-op) if the pin is not connected.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::EnumFilters

IFilterGraph Interface

Provides an enumerator for all filters in the graph.

HRESULT EnumFilters(
IEnumFilters ** *ppEnum*
);

Parameters

ppEnum
[out] Enumerator.

Return Values

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

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

Remarks

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

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::FindFilterByName

IFilterGraph Interface

Finds a filter that was added to the filter graph with a specific name.

HRESULT FindFilterByName(
LPCWSTR *pName*,
IBaseFilter ** *ppFilter*
);

Parameters

pName
[in, string] Name to search for.
ppFilter
[out] Pointer to an IBaseFilter interface on the found filter.

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 function fails and sets pointers to the *ppFilter* parameter to NULL if the name is not in this graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::Reconnect

IFilterGraph Interface

Disconnects this and the pin to which it connects and then reconnects it to the same pin. This allows the details of the connection, such as media type and allocator, to be renegotiated.

```
HRESULT Reconnect(
  IPin * ppin
  );
```

Parameters

ppin
[in] Pin to disconnect and reconnect.

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 performs its operation on a separate thread that will not hold any relevant locks. It can be called by a pin or filter to allow renegotiation of the connection. When a transform filter has its input connected, it must agree upon some media type. When the output is connected, it might discover that, to please both its upstream and downstream connections, it would have been better to have chosen a different media type for the upstream connection. The solution is to reconnect the input pin. The caller of this method should ensure (for example, by calling IPin::QueryAccept) that the resulting renegotiation will succeed, because the reconnection process is performed asynchronously and there is no mechanism for reporting or correcting errors.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::RemoveFilter

IFilterGraph Interface

Removes a filter from the graph.

```
HRESULT RemoveFilter(
  IBaseFilter * pFilter
  );
```

Parameters

pFilter
[in] Pointer to the filter to be removed from the graph.

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 graph implementation informs the filter that it is being removed by calling the IBaseFilter::JoinFilterGraph method with a NULL argument.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph::SetDefaultSyncSource

IFilterGraph Interface

Sets the default source of synchronization.

HRESULT SetDefaultSyncSource(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 used when no clock has been given to the filter graph, and the filter graph manager consequently must find a clock to use as the synchronization source. The filter graph manager first tries all filters, starting with renderers, to see if any filter exports a clock (by providing an IReferenceClock interface). The filter graph manager will choose the first filter that it finds, providing that filter is connected to an upstream source. If no connected filters are found, the first filter if **IReferenceClock** is used. Typically, this will be the audio rendering filter. If no filter exports a clock, the filter graph manager uses a system clock.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph2 Interface

The **IFilterGraph2** interface adds new functionality to the IGraphBuilder and IFilterGraph interfaces. It inherits from both interfaces and exposes all their methods. This interface also includes a method to add a source for a moniker, and an improved version of the IFilterGraph::Reconnect method. For this reason, you should usually use **IFilterGraph2** instead of the other two.

When to Implement

The filter graph implements this interface so it isn't intended that you implement it.

When to Use

Use this interface in applications that previously called IFilterGraph::Reconnect and in applications that need a source filter for a moniker.

Methods in Vtable Order

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

IFilterGraph methods	Description
AddSourceFilterForMoniker	Adds a source for a moniker.
ReconnectEx	Specifies a pin and a media type to reconnect with.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph2::AddSourceFilterForMoniker

IFilterGraph2 Interface

Adds a source for a moniker.

HRESULT AddSourceFilterForMoniker(
IMoniker **pMoniker*,
IBindCtx **pCtx*,
LPCWSTR *lpcwstrFilterName*,
IBaseFilter ***ppFilter* **);**

Parameters

pMoniker
[in] Pointer to an IMoniker interface.
pCtx
[in] Pointer to an IBindCtx bind context interface.
lpcwstrFilterName
[in] Pointer to the filter's name.
ppFilter
[out] Address of a pointer to an IBaseFilter interface.

Return Values

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

Remarks

When adding a source filter for the given moniker to the graph, the COM

IMoniker::BindToStorage member function will query for an IStream interface. If this fails, IMoniker::BindToObject will try to retrieve an IBaseFilter interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterGraph2::ReconnectEx

IFilterGraph2 Interface

Specifies a pin and a media type to reconnect with.

HRESULT ReconnectEx(
IPin * *ppin*,
const AM_MEDIA_TYPE *pmt **);**

Parameters

ppin
[in] Pin to disconnect and reconnect.
pmt
[in] Media type to reconnect with. Specify NULL to use the existing media type.

Return Values

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

Remarks

Like the IFilterGraph::Reconnect method, the **ReconnectEx** method schedules a reconnection of the pin with the pin it is currently connected to. By specifying a media type when this method is called, the pins don't have to check what type they were originally connected with or enumerate possible new types. This makes the reconnection more likely to succeed.

See Also

IFilterGraph::Reconnect

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterInfo Interface

IFilterInfo is an interface that manages information about a filter and provides access to the filter and to the IPinInfo interfaces representing the pins on the filter. It is essentially an IBaseFilter interface that can be accessed through Automation. This was created to provide access to the **IBaseFilter** methods from Microsoft® Visual Basic® applications without incurring the overhead of Automation in the **IBaseFilter** interface itself.

When to Implement

This interface is implemented by the filter graph manager for use by Automation client applications, such as Microsoft Visual Basic.

When to Use

Use this interface from an application to retrieve information about a filter and to retrieve individual pin objects in the filter or a collection of all pin objects belonging to the filter. This can be used when adding filters to a filter graph and connecting pins together.

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.

IFilterInfo methods	Description
FindPin	Locates a pin and returns an IPinInfo interface.
get_Name	Retrieves the filter name.
get_VendorInfo	Retrieves a string containing optional information supplied by a vendor about the specified filter.
get_Filter	Retrieves the IBaseFilter interface for the filter.
get_Pins	Retrieves an IAMCollection interface which provides access to the IPinInfo interfaces for this filter.
get_IsFileSource	Determines if the filter is a file source filter.
get_Filename	Retrieves the file name associated with the source filter.

put_Filename Sets the file name containing the media source.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterInfo::FindPin

IFilterInfo Interface

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

```
HRESULT FindPin(
  BSTR strPinID,
  IDispatch **ppUnk
  );
```

Parameters

strPinID
[in] String pin identifier.
ppUnk
[out] IPinInfo interface.

Return Values

Returns an HRESULT value.

Remarks

This method corresponds to the IBaseFilter::FindPin method. This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::get_Filename

IFilterInfo Interface

Retrieves the file name associated with the source filter.

HRESULT get_Filename(
BSTR **pstrFilename*
);

Parameters

pstrFilename
[out, retval] File name containing the source media.

Return Values

Returns an HRESULT value.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::get_Filter

IFilterInfo Interface

Retrieves the IBaseFilter interface of the filter.

HRESULT get_Filter(
IUnknown ***ppUnk*
);

Parameters

ppUnk
[out, retval] IBaseFilter interface of the filter represented by IFilterInfo.

Return Values

Returns an HRESULT value.

Remarks

The object that implements IFilterInfo is a wrapper and is not the same COM object as the filter itself. Thus a call to **IFilterInfo::QueryInterface** for IBaseFilter will fail. The

IFilterInfo::get_Filter method allows an application to obtain the filter object itself. This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::get_IsFileSource

IFilterInfo Interface

Determines if the filter is a file source filter.

HRESULT get_IsFileSource(
LONG *pbIsSource*
);

Parameters

pbIsSource
[out, retval] Returned Boolean value.

Return Values

Returns an HRESULT value. Returns OATRUE if filter is a file source filter; otherwise, returns OAFALSE.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::get_Name

IFilterInfo Interface

Retrieves the filter name.

HRESULT get_Name(
BSTR **strName*
);

Parameters

strName
[out, retval] Name of the filter.

Return Values

Returns an HRESULT value.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::get_Pins

IFilterInfo Interface

Retrieves an IAMCollection interface, which provides access to the IPinInfo interfaces for the pins on this filter.

HRESULT get_Pins(
IDispatch ***ppUnk*
);

Parameters

ppUnk
[out, retval] IAMCollection interface.

Return Values

Returns an HRESULT value.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations. Visual Basic applications can enumerate the IPinInfo interfaces in the returned IAMCollection object by using the `For Each ...Next`

syntax.

Previous Home Topic Contents Index Next

IFilterInfo::get_VendorInfo

IFilterInfo Interface

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

HRESULT get_VendorInfo(
BSTR **strVendorInfo*
);

Parameters

strVendorInfo
[out, retval] String containing vendor information.

Return Values

Returns an HRESULT value.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterInfo::put_Filename

IFilterInfo Interface

Sets the file name containing the media source.

HRESULT put_Filename(
BSTR *strFilename*

```
);
```

Parameters

strFilename
[in] Name of the file for the source filter to read from.

Return Values

Returns an HRESULT value.

Remarks

This method is exposed for use by Automation clients and is not expected to be used by C or C++ applications because of performance limitations.

Previous Home Topic Contents Index Next

IFilterMapper Interface

The **IFilterMapper** interface is an abstraction that represents registered information about filters. This allows properties of filters to be looked up during loading.

When to Implement

This interface is implemented on the filter mapper and is not intended to be implemented by developers.

When to Use

This interface is used by filters to register and unregister themselves. This is handled in the base classes by the CBaseFilter::Register and CBaseFilter::Unregister member functions. It is also used by the filter graph manager to look up filters and determine their characteristics when building a filter graph to render a given media type.

Methods in Vtable Order

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

IFilterMapper methods	Description
RegisterFilter	Records the details of a filter in the registry.
RegisterFilterInstance	Registers an identifiable instance of a filter.
RegisterPin	Records the details of a pin in the registry.
RegisterPinType	Adds a type for the pin to the registry.
UnregisterFilter	Deletes a filter from the registry.
UnregisterFilterInstance	Deletes an identifiable instance of a filter.
UnregisterPin	Deletes a pin from the registry.
EnumMatchingFilters	Finds all filters matching specific requirements.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::EnumMatchingFilters

IFilterMapper Interface

Provides an enumerator that enumerates registered filters that meet specified requirements.

HRESULT EnumMatchingFilters(
IEnumRegFilters ** *ppEnum*,
DWORD *dwMerit*,
BOOL *bInputNeeded*,
CLSID *clsInMaj*,
CLSID *clsInSub*,
BOOL *bRender*,
BOOL *bOutputNeeded*,
CLSID *clsOutMaj*,
CLSID *clsOutSub*
);

Parameters

ppEnum
[out] Enumerator returned.
dwMerit
[in] Enumerate only filters with at least this merit.
bInputNeeded
TRUE if there must be at least one input pin.
clsInMaj
[in] Input major type required. Set to GUID_NULL if you do not care.
clsInSub
[in] Input subtype required. Set to GUID_NULL if you do not care.
bRender

[in] Option that specifies if the input must be rendered by this filter.

bOutputNeeded

TRUE if there must be at least one output pin.

clsOutMaj

[in] Output major type required. Set to GUID_NULL if you do not care.

clsOutSub

[in] Output subtype required. Set to GUID_NULL if you do not care.

Return Values

Returns an HRESULT value.

Remarks

Set the *ppEnum* parameter to be an enumerator for filters matching the requirements. For a description of merit values for the *dwMerit* parameter, see the IFilterMapper::RegisterFilter method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::RegisterFilter

IFilterMapper Interface

Adds a filter to the registry; the filter can then be enumerated.

```
HRESULT RegisterFilter(
  CLSID clsid,
  LPCWSTR Name,
  DWORD dwMerit
  );
```

Parameters

clsid

[in] Globally unique identifier (GUID) of the filter.

Name

[in] Descriptive name for the filter.

dwMerit

[in] Position in the order of enumeration. Filters with higher merit are enumerated first.

Return Values

Returns an HRESULT value.

Remarks

The merit (as defined by the *dwMerit* parameter) controls the order in which the filter graph manager tries filters when performing an operation as a result of a call to IGraphBuilder::Connect, IGraphBuilder::Render, or IGraphBuilder::RenderFile. The filter graph manager finds all filters registered with the correct media type and then tries the one with the highest merit, using other criteria in the registration to choose between filters with equal merit.

The following predefined values exist for the *dwMerit* parameter. Other values, such as MERIT_NORMAL-1, can be used. Using the formula (MERIT_NORMAL+MERIT_UNLIKELY)/2 to calculate the merit value is advisable, because it leaves some low-order bits available for making even finer distinctions.

0x900000	Hardware renderer filter. Filters with this merit value are tried first.
MERIT_PREFERRED (0x800000)	Filter, such as a renderer, that is likely to complete the operation directly.
0x680000	MPEG decompression filter. These are tried before AVI decompressors because the latter require more time to determine if they work in the filter graph.
MERIT_NORMAL (0x600000)	Filter that may contribute to the completion of a connection. AVI decompression filters and splitter transform filters are registered to this value.
MERIT_UNLIKELY (0x400000)	Filter that may contribute to the completion of a connection (a color space conversion filter, for example). The filter graph manager uses this filter only if other options have failed. Register source filters with this value.
MERIT_DO_NOT_USE (0x200000)	Filter that will never contribute to the completion of a connection. Filters registered with this value (or less) will never be tried by the filter graph manager when automatically building a filter graph. Use this to register filters that must be added explicitly, either as part of a predefined filter graph or by adding them using IFilterGraph::AddFilter. Examples include tee filters or effects filters.
MERIT_HW_COMPRESSOR (0x100050)	Hardware compressor filter. Filters registered with this value will never be tried by the filter graph manager when automatically building a filter graph.
MERIT_SW_COMPRESSOR (0x100000)	Software compressor filter. Filters registered with this value will never be tried by the filter graph manager when automatically building a filter graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::RegisterFilterInstance

IFilterMapper Interface

Registers an identifiable instance of a filter.

HRESULT RegisterFilterInstance(
CLSID *clsid*,
LPCWSTR *Name*,
CLSID **MRId*
);

Parameters

clsid
[in] Globally unique identifier (GUID) of the filter.
Name
[in] Descriptive name of the instance.
MRId
[out] Returned media resource ID. This parameter is a locally unique identifier for this instance of this filter.

Return Values

Returns an HRESULT value.

Remarks

This method handles cases such as when two similar sound cards that are driven by the same driver are available, and it is necessary to choose which card will emit the sound. This is not needed if there is only one instance of the filter (such as when there is only one sound card in the computer), or if all instances of the filter are equivalent.

The filter itself must have already been registered.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::RegisterPin

IFilterMapper Interface

Records the details of the pin in the registry.

```
HRESULT RegisterPin(
  CLSID Filter,
  LPCWSTR Name,
  BOOL bRendered,
  BOOL bOutput,
  BOOL bZero,
  BOOL bMany,
  CLSID ConnectsToFilter,
  LPWSTR ConnectsToPin
  );
```

Parameters

Filter
[in] Globally unique identifier (GUID) of the filter.

Name
[in] Name of the pin. This should be unique within the filter. It has no significance other than to indicate type information. Note that pin names longer than 99 characters should not be used, because this causes filter enumeration problems.

bRendered
[in] Set to TRUE if the filter renders this input; otherwise, set to FALSE.

bOutput
[in] Set to TRUE if this is an output pin; otherwise, set to FALSE.

bZero
[in] If the filter can have zero instances of this pin, set to TRUE; otherwise, set to FALSE. For example, a decompression filter might choose to not create a sound output pin for a movie without a sound track.

bMany
[in] If the filter can have many instances of this pin, set to TRUE; otherwise, set to FALSE. For example, a mixer might have multiple instances of its input pin.

ConnectsToFilter
[in] Reserved. Must be NULL. (This is intended for filters such as system-wide mixers that have connections outside the filter graph. It is not yet implemented.)

ConnectsToPin
[in] Reserved. Must be NULL.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::RegisterPinType

IFilterMapper Interface

Registers this pin type.

HRESULT RegisterPinType(
CLSID *clsFilter*,
LPCWSTR *strName*,
CLSID *clsMajorType*,
CCLSID *clsSubType*
);

Parameters

clsFilter
Class identifier (CLSID) of the filter to which the pin belongs.
strName
Name by which it is known.
clsMajorType
Major type of the media sample supported by this pin class.
clsSubType
Subtype of the media sample supported by this pin class.

Return Values

Returns an HRESULT value.

Remarks

The *clsMajorType* and *clsSubType* parameters specify the media type of the pin and correspond to the AM_MEDIA_TYPE structure's majortype and subtype members, respectively.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper::UnregisterFilter

IFilterMapper Interface

Removes the registration of this filter from the registry.

HRESULT UnregisterFilter(
CLSID *Filter*
);

Parameters

Filter
[in] Globally unique identifier (GUID) of the filter.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IFilterMapper::UnregisterFilterInstance

IFilterMapper Interface

Removes the registration of this filter instance from the registry.

HRESULT UnregisterFilterInstance(
CLSID *MRId*
);

Parameters

MRId
[in] Media resource identifier of this instance.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IFilterMapper::UnregisterPin

IFilterMapper Interface

Removes the registration of this pin from the registry.

HRESULT UnregisterPin(
CLSID *Filter*,

LPCWSTR *Name*
);

Parameters

Filter
[in] Globally unique identifier (GUID) of the filter that this pin is part of.
Name
[in] Name of the pin.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IFilterMapper2 Interface

IFilterMapper2 is an interface for registering and locating DirectShow filters with greater flexibility than IFilterMapper allowed. Applications and filters should use **IFilterMapper2** instead of **IFilterMapper**, although both interfaces can find filters registered with the other interface.

One major change from IFilterMapper to **IFilterMapper2** is that **IFilterMapper2** provides support for filter categories. A filter can appear in one or more categories (for example, Video Compressors) to restrict the search space. The RegisterFilter method takes a category, and the EnumMatchingFilters method searches across categories.

Other changes include:

1. Quicker and easier enumeration of hardware devices such as WDM/PnP
2. Registration in one step (previously you had to register pins and media types with separate calls)
3. Register and search by mediums (see Kernel Streaming in the NT DDK)

When to Implement

This interface is implemented on the filter graph and is not intended to be implemented by developers.

When to Use

Applications and filters should use IFilterMapper2 when they need to register or unregister filters. **IFilterMapper2** should be used instead of IFilterMapper.

Methods in Vtable Order

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

IFilterMapper2 methods	Description
CreateCategory	Adds a new category to the list of filter categories (CLSID_ActiveMovieCategories).
UnregisterFilter	Removes the registration of the specified filter from the registry.
RegisterFilter	Registers a filter, pins, and media types under a category.
EnumMatchingFilters	Provides an enumerator that enumerates registered filters that meet specified requirements.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper2::CreateCategory

IFilterMapper2 Interface

Adds a new category to the list of filter categories (CLSID_ActiveMovieCategories).

HRESULT CreateCategory(
  REFCLSID *clsidCategory*,
  DWORD *dwCategoryMerit*,
  LPCWSTR *Description* **);**

Parameters

clsidCategory
[in] Name of the new filter category.
dwCategoryMerit
[in] Merit value of the category. Categories with higher merit are enumerated first.
Description
[in] Descriptive name for the category.

Return Values

Returns S_OK on success; HRESULT_FROM_WIN32 on failure.

Remarks

The graph builder initially skips all categories with merit less than DO_NOT_USE to speed up

the IGraphBuilder::RenderFile method. Categories of filters that should not be considered for playback should be marked DO_NOT_USE.

A filter can appear in one or more categories (for example, Video Compressors) to restrict the search space.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper2::EnumMatchingFilters

IFilterMapper2 Interface

Provides an enumerator that enumerates registered filters that meet specified requirements.

HRESULT EnumMatchingFilters(
IEnumMoniker ***ppEnum*,
DWORD *dwFlags*,
BOOL *bExactMatch*,
DWORD *dwMerit*,
BOOL *bInputNeeded*,
DWORD *cInputTypes*,
const GUID **pInputTypes*,
const REGPINMEDIUM **pMedIn*,
const CLSID **pPinCategoryIn*,
BOOL *bRender*,
BOOL *bOutputNeeded*,
DWORD *cOutputTypes*,
const GUID **pOutputTypes*,
const REGPINMEDIUM **pMedOut*,
const CLSID **pPinCategoryOut*,**);**

Parameters

ppEnum
[out] Enumerator returned.
dwFlags
[in] Currently reserved, specify zero.
bExactMatch
[in] Specify TRUE to indicate wildcards not allowed; FALSE indicates wildcards allowed.
dwMerit
[in] Enumerate only filters with at least the specified merit value.
bInputNeeded
[in] TRUE if there must be at least one input pin.
cInputTypes
[in] Number of input types to match.

pInputTypes
[in] Input major types and subtype required. Set to GUID_NULL if you do not care.
pMedIn
[in] Input medium. Set to NULL if not needed.
pPinCategoryIn
[in] Input pin category. Set to NULL if not needed.
bRender
[in] Option that indicates if the specified filter must render the input.
bOutputNeeded
[in] TRUE if there must be at least one output pin.
cOutputTypes
[in] Number of output types to match.
pOutputTypes
[in] Output major type and subtype required. Set to GUID_NULL if you do not care.
pMedOut
[in] Output medium. Set to NULL if not needed.
pPinCategoryOut
[in] Output pin category. Set to NULL if not needed.

Return Values

Returns S_OK if successful or E_FAIL upon failure.

Remarks

If a pin hasn't registered any media types, this method will not consider a match for the media type passed in for the *pInputTypes* or *pOutputTypes* parameters.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper2::RegisterFilter

IFilterMapper2 Interface

Registers a filter, pins, and media types under a category.

HRESULT RegisterFilter(
REFCLSID *clsidFilter*,
LPCWSTR *Name*,
IMoniker ***ppMoniker*,
const CLSID **pclsidCategory*,
const OLECHAR **szInstance*,
const REGFILTER2 **prf2* **);**

Parameters

clsidFilter
[in] GUID of the filter. CoCreateInstance will be called with this **GUID** when the filter is instantiated.
Name
[in] Descriptive name for the filter.
ppMoniker
[in, out] Address of a pointer to a device moniker that determines where this filter's data will be written. This parameter will be set to NULL upon return.
pclsidCategory
[in] Category of the filter to register.
szInstance
[in] Unique identifier for the filter (can be the friendly name or the filter CLSID).
prf2
[in] Pointer to a REGFILTER2 structure containing merit and pin information.

Return Values

Returns one of the following values:

Value	Meaning
VFW_E_BAD_KEY	Couldn't get registry key.
HRESULT_FROM_WIN32	Failure.
NOERROR	Success.

Remarks

Specify the moniker in the *ppMoniker* parameter if you are registering a filter for a WDM/PnP (Windows Driver Model/Plug and Play) device. If *ppMoniker* is non-null, the moniker returned can be used to write additional private values in the property bag.

The *pclsidCategory* parameter defaults to CLSID_ActiveMovieFilters if NULL is passed in.

Set *ppMoniker* to NULL if you don't want to provide or receive the moniker.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFilterMapper2::UnregisterFilter

IFilterMapper2 Interface

Removes the registration of the specified filter from the registry.

HRESULT UnregisterFilter(
const CLSID **pclsidCategory*,
const OLECHAR **szInstance*,
REFCLSID *Filter* **);**

Parameters

pclsidCategory
[in] Name of the category that the filter falls under.
szInstance
[in] Name of the filter you want to remove.
Filter
[in] Globally unique identifier (GUID) of the filter.

Return Values

Returns S_OK on success; HRESULT_FROM_WIN32 on failure.

Remarks

If *szInstance* is NULL, this method uses the filter **GUID** passed in *Filter*.

This method might return an error if the filter was not registered.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFullScreenVideo Interface

This interface allows an application or plug-in distributor to control a full-screen renderer. The Microsoft® DirectShow™ full-screen renderer supports this interface. When connected, a renderer should load the display modes that are available. The number of modes available can be obtained through IFullScreenVideo::CountModes. Information on each individual mode is then available by calling IFullScreenVideo::GetModeInfo and IFullScreenVideo::IsModeAvailable. An application can enable and disable any mode by calling the IFullScreenVideo::SetEnabled method. The current value can be queried by calling IFullScreenVideo::IsModeEnabled. An application can enable or disable any mode by calling the **IFullScreenVideo::SetEnabled** flag with OATRUE or OAFALSE (not C/C++ TRUE and FALSE values).

The DirectShow full-screen renderer can function only when it is the foreground active window. If the user tries to switch to another application while in full-screen mode, the video will be hidden. The renderer does this by minimizing the window it is actually drawing the video in (although the use of a window is transparent to the user). Maximizing the window restores the

video.

The filter graph manager uses the full-screen renderer to help implement the full-screen property that can be set through IVideoWindow::put_FullScreenMode. When used in this manner, the expected behavior is to switch seamlessly between full-screen mode and then back into normal window playback when the user presses ESC or other escape sequences. In this case, the renderer is required to hide its window rather than minimize it when switching away from it. The renderer can be made to hide rather than minimize (the default action) by calling IFullScreenVideo::HideOnDeactivate.

When to Implement

Implement this interface if you are writing an alternate full-screen video renderer. The Microsoft full-screen video renderer implements this interface by default.

When to Use

Use this interface from any application that must interact with the full-screen video renderer in order to set or determine video modes or other full-screen renderer properties.

Methods in Vtable Order

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

IFullScreenVideo methods	Description
CountModes	Retrieves the number of modes available on the full-screen renderer.
GetModeInfo	Retrieves information about a specified mode.
GetCurrentMode	Retrieves the video mode currently in effect.
IsModeAvailable	Determines if a specified video mode is available.
IsModeEnabled	Determines if a specified video mode is enabled.
SetEnabled	Enables and disables video modes.
GetClipFactor	Retrieves the current clip loss factor setting.
SetClipFactor	Specifies the video mode based on the maximum image area that will be lost.
SetMessageDrain	Specifies a window that will receive window messages sent to the renderer.
GetMessageDrain	Retrieves the window set to receive window messages sent to the renderer.
SetMonitor	Sets the monitor in use (for use with multiple-monitor implementations).
GetMonitor	Retrieves the monitor in use.
HideOnDeactivate	Hides the window when it is deactivated.
IsHideOnDeactivate	Retrieves the state of the **HideOnDeactivate** property.
SetCaption	Sets the caption associated with the full-screen window.
GetCaption	Retrieves the caption associated with the full-screen window.
SetDefault	Sets the default for all full-screen video renderer properties.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFullScreenVideo::CountModes

IFullScreenVideo Interface

Retrieves the number of modes available on the full-screen renderer.

HRESULT CountModes(
long **pModes*
);

Parameters

pModes
[out] Returned mode count.

Return Values

Returns an HRESULT value.

Remarks

The Microsoft DirectShow full-screen renderer supports eight modes (320 x 200 x 8/16 bit, 320 x 240 x 8/16, 640 x 400 x 8/16, and 640 x 480 x 8/16). The lower-resolution modes are always chosen over the higher modes. An application can enable and disable specific modes to more accurately specify which ones are available for use. A slightly less fine-grained mechanism to affect the selected mode is available through the clip loss factor.

Previous Home Topic Contents Index Next

IFullScreenVideo::GetCaption

IFullScreenVideo Interface

Retrieves the caption associated with the full-screen window.

HRESULT GetCaption(
BSTR **pstrCaption*
);

Parameters

pstrCaption
[out] Retrieved caption.

Return Values

Returns an HRESULT value.

Remarks

The Microsoft DirectShow full-screen renderer will show itself as an icon when it is deactivated (you can deactivate it by pressing ALT+TAB to switch away from it). The text caption for the icon can be set through this method. If the renderer is supposed to hide itself when deactivated, this method will succeed, although it will have no use because the window will never be shown as an icon.

Previous Home Topic Contents Index Next

IFullScreenVideo::GetClipFactor

IFullScreenVideo Interface

Retrieves the current clip loss factor setting.

HRESULT GetClipFactor(
long **pClipFactor*
);

Parameters

pClipFactor
[out] Maximum allowable amount of the image to lose.

Return Values

Returns an HRESULT value.

Remarks

For a description of the clip loss factor setting, see IFullScreenVideo::SetClipFactor.

IFullScreenVideo::GetCurrentMode

IFullScreenVideo Interface

Retrieves the video mode currently in effect.

```
HRESULT GetCurrentMode(
  long *pMode
  );
```

Parameters

pMode
[out] Retrieved full-screen video mode.

Return Values

Returns an HRESULT value.

Remarks

When the full-screen video renderer is connected, it chooses a display mode to use for video playback. The selected mode can be retrieved through this method. The details for that mode can then be retrieved by using IFullScreenVideo::GetModeInfo.

For a list of available modes for the Microsoft DirectShow full-screen video renderer, see IFullScreenVideo::SetEnabled.

IFullScreenVideo::GetMessageDrain

IFullScreenVideo Interface

Retrieves the window set to receive window messages sent to the renderer.

HRESULT GetMessageDrain(
HWND **hwnd*
);

Parameters

hwnd
[out] Window handle of the window specified as the message drain.

Return Values

Returns an HRESULT value.

Remarks

The full-screen video renderer posts all mouse and keyboard messages it receives to the window designated as a message drain, with the message parameters untranslated. The exact list of messages passed is the same as for IVideoWindow::put_MessageDrain. An application can use this to implement hot-key support for full-screen video. For example, it might watch for the CTRL+P key combination as a cue to pause the video.

Previous Home Topic Contents Index Next

IFullScreenVideo::GetModeInfo

IFullScreenVideo Interface

Retrieves information about a specified mode.

HRESULT GetModeInfo(
long *Mode*,
long **pWidth*,
long **pHeight*,
long **pDepth*
);

Parameters

Mode
[in] Specified mode for which to retrieve information.
pWidth
[out] Width in pixels of the mode's image display.
pHeight
[out] Height in pixels of the mode's image display.
pDepth

[out] Number of color bits per pixel.

Return Values

Returns an HRESULT value.

Remarks

This method returns the height, width, and color depth for a given mode that the renderer has available. The number of display modes available can be retrieved through IFullScreenVideo::CountModes.

For a list of modes available to the Microsoft DirectShow full-screen video renderer, see IFullScreenVideo::SetEnabled.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFullScreenVideo::GetMonitor

IFullScreenVideo Interface

Retrieves the monitor type in use.

```
HRESULT GetMonitor(
  long *Monitor
  );
```

Parameters

Monitor
[out] Monitor currently in use.

Return Values

Returns NOERROR.

Remarks

The Microsoft DirectShow full-screen renderer always returns zero (the primary monitor).

Previous Home Topic Contents Index Next

IFullScreenVideo::HideOnDeactivate

IFullScreenVideo Interface

Hides the window icon when the full-screen window is deactivated.

HRESULT HideOnDeactivate(
long *Hide*
);

Parameters

Hide
[in] Set to OATRUE to hide the video window icon when deactivated; set to OAFALSE to display the icon.

Return Values

Returns NOERROR if successful and E_INVALIDARG if *Hide* is invalid.

Remarks

The default setting for the Microsoft DirectShow full-screen renderer is OATRUE.

The full-screen renderer can function only when it is the foreground active window. If the user tries to switch to another application while in full-screen mode, the video will be hidden. The renderer does this by minimizing the window it is actually drawing the video in (although the use of a window is transparent to the user). Maximizing the window restores the video. If this property is set, the window will be hidden rather than minimized.

Previous Home Topic Contents Index Next

IFullScreenVideo::IsHideOnDeactivate

IFullScreenVideo Interface

Retrieves the state of the IFullScreenVideo::HideOnDeactivate property.

HRESULT IsHideOnDeactivate(void);

Return Values

Returns S_OK if **HideOnDeactivate** is set to OATRUE or S_FALSE if it is set to OAFALSE.

Previous Home Topic Contents Index Next

IFullScreenVideo::IsModeAvailable

IFullScreenVideo Interface

Determines if a specified video mode is available.

HRESULT IsModeAvailable(
long *Mode*
);

Parameters

Mode
[in] Full-screen mode to check.

Return Values

Returns S_OK if the video mode is available or S_FALSE if it isn't.

Remarks

The display modes supported by the Microsoft DirectShow full-screen renderer are different than the actual modes available on any given display card. The application should call this method only when the renderer is connected, because until then the renderer might not have allocated the resources it needs to make this information available. Even if a mode is available, it will not necessarily be used for video playback; the mode must also be compatible with the filters in the filter graph.

Previous Home Topic Contents Index Next

IFullScreenVideo::IsModeEnabled

IFullScreenVideo Interface

Determines if a specified mode is enabled.

HRESULT IsModeEnabled(
long *Mode*
);

Parameters

Mode
[in] Full-screen video mode to check.

Return Values

Returns S_OK if the video mode is enabled or S_FALSE if it isn't.

Remarks

You can use this method to enable and disable specific display modes supported by the renderer. Even if a mode is enabled, it will not necessarily be used for video playback; the mode must also be compatible with the filters in the filter graph.

For a list of available modes for the Microsoft DirectShow full-screen video renderer, see IFullScreenVideo::SetEnabled.

See Also

IFullScreenVideo::IsModeAvailable

Previous Home Topic Contents Index Next

IFullScreenVideo::SetCaption

IFullScreenVideo Interface

Sets the title associated with the full-screen window.

HRESULT SetCaption(
BSTR *strCaption*
);

Parameters

strCaption
[in] String containing the caption.

Return Values

Returns an HRESULT value.

Previous Home Topic Contents Index Next

IFullScreenVideo::SetClipFactor

IFullScreenVideo Interface

Specifies the video mode based on the maximum data that will be lost.

```
HRESULT SetClipFactor(
  long ClipFactor
  );
```

Parameters

ClipFactor
[in] Maximum allowable amount of the image to lose.

Return Values

Returns an HRESULT value.

Remarks

Clip loss factor is a means of enabling full-screen modes that is more generic and easier for applications to use. This method defines the amount of video that can be lost when deciding which display mode to use. For example, assuming the decoder cannot compress the video, playing an MPEG file (say 352 x 288 pixels) into a 320 x 200 display will lose about 25 percent of the image. The clip loss factor specifies the upper range permissible. To allow typical QCIF-sized MPEG video (352 x 288 pixels) to be played in a 320 x 200 display mode, it defaults to 25 percent.

Previous Home Topic Contents Index Next

IFullScreenVideo::SetDefault

IFullScreenVideo Interface

Sets the default full-screen video renderer settings.

HRESULT SetDefault(void);

Return Values

Returns an HRESULT value.

Remarks

The properties set through this interface apply only to the current renderer instance. They can, however, be made the global default by calling this interface method.

Previous Home Topic Contents Index Next

IFullScreenVideo::SetEnabled

IFullScreenVideo Interface

Enables and disables video modes.

HRESULT SetEnabled(
long *Mode*,
long *bEnabled*
);

Parameters

Mode
[in] Mode to be enabled or disabled (see the following table).
bEnabled
[out] Can be set to one of the following values.

Value	Meaning
OATRUE	Enable mode.
OAFALSE	Disable mode.

Return Values

Returns an HRESULT value.

Remarks

Available DirectShow modes are defined as follows. The ordering of these modes is subject to change, so use IFullScreenVideo::CountModes and IFullScreenVideo::GetModeInfo interface methods to enumerate modes.

Mode	Width	Height	RGB depth (in bits)
0	320	200	16
1	320	200	8
2	320	240	16
3	320	240	8
4	640	400	16
5	640	400	8
6	640	480	16
7	640	480	8
8	800	600	16
9	800	600	8
10	1024	768	16
11	1024	768	8
12	1152	864	16
13	1152	864	8
14	1280	1024	16
15	1280	1024	8

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IFullScreenVideo::SetMessageDrain

IFullScreenVideo Interface

Specifies a window that will receive window messages sent to the renderer.

```
HRESULT SetMessageDrain(
  HWND hwnd
  );
```

Parameters

hwnd
[in] Window handle of the message drain window.

Return Values

Returns an HRESULT value.

Remarks

The full-screen video renderer posts all mouse and keyboard messages it receives to the window designated as a message drain, with the message parameters untranslated. The exact list of messages passed is the same as for IVideoWindow::put_MessageDrain. An application can use this to implement hot-key support for full-screen video. For example, it might watch for the CTRL+P key combination as a cue to pause the video.

Previous Home Topic Contents Index Next

IFullScreenVideo::SetMonitor

IFullScreenVideo Interface

Sets the monitor type in use.

HRESULT SetMonitor(
long *Monitor*
);

Parameters

Monitor
[in] Monitor currently in use.

Return Values

Returns an HRESULT value.

Remarks

Setting this method to anything but the primary display (0) will return an error. In future versions of Microsoft DirectShow, the renderer might allow applications to select the monitor on which to play back the full-screen video.

Previous Home Topic Contents Index Next

IGraphBuilder Interface

The **IGraphBuilder** interface allows applications to call upon the filter graph manager to attempt to build a complete filter graph, or parts of a filter graph given only partial information, such as the name of a file or the interfaces of two separate pins. The filter mapper looks up filters in the registry to configure the filter graph in a meaningful way.

IGraphBuilder inherits from the IFilterGraph interface and exposes all its methods. For this reason, **IFilterGraph** should normally not be used directly.

When to Implement

This interface is implemented on the filter graph manager and is not intended for implementation by developers.

When to Use

Applications use this interface to create a filter graph, add filters to or remove filters from a filter graph, enumerate all the filters in a filter graph, and force connections when adding a filter. Filters typically use the interface to reconnect pins during the connection and negotiation process of building a filter graph.

Methods in Vtable Order

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

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

IGraphBuilder methods	Description
Connect	Connects two IPin objects. If they will not connect directly, this method connects them with intervening transforms.
Render	Adds a chain of filters to this output pin to render it.
RenderFile	Builds a filter graph that renders the specified file.
AddSourceFilter	Adds a source filter to the filter graph for a specific file. The IGraphBuilder::RenderFile method calls this to find the source filter.
SetLogFile	Sets the log file into which actions taken in attempting to perform an operation are logged.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphBuilder::AddSourceFilter

IGraphBuilder Interface

Adds a source filter to the filter graph for a specific file.

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

Parameters

lpwstrFileName
[in] Pointer to the file.
lpwstrFilterName
[in] Name to give the source filter when it is added.
ppFilter
[out] Pointer to an IBaseFilter interface on the filter that was added.

Return Values

Returns an HRESULT value.

Remarks

This method allows you to obtain and retain more control over building the rest of the graph. For example, you can use the IFilterGraph::AddFilter method to add a renderer of your choice, and then use the IGraphBuilder::Connect method to connect the two filters. The IBaseFilter interface exposed by the source filter is returned in the *ppFilter* parameter, and the reference is already added by the IUnknown::AddRef method. The *lpwstrFilterName* parameter is used to allow the filter to be identified by this name in this filter graph. For more information, see FindFilterByName.

It is the application's responsibility to find the output pin of the added source filter in order to build the rest of the filter graph, which can be done by calling IGraphBuilder::Render on the output pin, to build the entire filter graph automatically, or by adding and connecting filters individually. Note that, when adding filters individually, the asynchronous file reader source filter and the URL moniker source filter do not parse the data, so the output pins of these source filters can be connected only to a parser filter, such as the MPEG splitter filter.

Note that the IGraphBuilder::RenderFile method adds the same source filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphBuilder::Connect

IGraphBuilder Interface

Connects the two pins, using intermediates if necessary.

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

Parameters

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

Return Values

Returns an HRESULT value.

Remarks

This method connects these two pins directly or indirectly, using transform filters if necessary. The method either succeeds or leaves the filter graph unchanged. The filter graph manager attempts a direct connection. If that fails, it attempts to use any available transforms provided by filters that are already in the filter graph. (It enumerates these in an arbitrary order.) If that fails, it attempts to find filters from the registry to provide a transform. These will be tried in order of merit.

Previous Home Topic Contents Index Next

IGraphBuilder::Render

IGraphBuilder Interface

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

HRESULT Render(
IPin * *ppinOut*
);

Parameters

ppinOut
[in] Output pin.

Return Values

Returns an HRESULT value, which can include one of the following:
VFW_S_AUDIO_NOT_RENDERED
VFW_S_DUPLICATE_NAME
VFW_S_PARTIAL_RENDER
VFW_S_RPZA
VFW_S_VIDEO_NOT_RENDERED

Remarks

This method connects this output pin directly or indirectly to a filter or filters that will render it, using transform filters as intermediary filters if necessary. Filters are tried in the same order as for the IGraphBuilder::Connect method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphBuilder::RenderFile

IGraphBuilder Interface

Builds a filter graph that renders the specified file.

HRESULT RenderFile(
LPCWSTR *lpwstrFile*,
LPCWSTR *lpwstrPlayList*
);

Parameters

lpwstrFile
[in] Name of the file containing the data to be rendered.
lpwstrPlayList
[in] Playlist name. Reserved; must be NULL. (This parameter is currently unimplemented.)

Return Values

Returns an HRESULT value, which can include one of the following:
VFW_S_AUDIO_NOT_RENDERED
VFW_S_DUPLICATE_NAME
VFW_S_PARTIAL_RENDER
VFW_S_RPZA
VFW_S_VIDEO_NOT_RENDERED

Remarks

If the *lpwstrPlayList* parameter is NULL, this method would use the default playlist, which typically renders the entire file.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphBuilder::SetLogFile

IGraphBuilder Interface

Sets the file into which actions taken in attempting to perform an operation are logged.

HRESULT SetLogFile(
HANDLE *hFile*
);

Parameters

hFile
Handle to the log file.

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 *hFile* parameter must be an open file handle. After calling this method with a valid file handle, actions taken by IGraphBuilder methods when attempting to build a filter graph are logged to this file. This is intended to help you determine the cause of any failure to automatically build a filter graph.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphVersion Interface

The **IGraphVersion** interface is provided by the filter graph manager to let other objects, especially plug-in distributors and the Graphedt.exe tool, know when the graph has changed.

When to Implement

This interface is implemented by the filter graph manager.

When to Use

Use this interface if your application or plug-in distributor must know when filters have been added, deleted, or reconnected.

Methods in Vtable Order

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

IGraphVersion methods	Description
QueryVersion	Returns the current graph version number.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IGraphVersion::QueryVersion

IGraphVersion Interface

Returns the current graph version number.

HRESULT QueryVersion(
LONG* *pVersion*
);

Parameters

pVersion
Current graph version.

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 version number is incremented every time there is a change in the set of filters in the graph or in their connections. If the version number has changed since the last enumeration, the graph must be re-enumerated.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IKsPropertySet Interface

The **IKsPropertySet** interface enables you to set and retrieve device properties.

When to Implement

Implement this interface on any pin to expose its properties and to enable its properties to be changed.

When to Use

Use this interface in your application or filter to access device properties.

Methods in Vtable Order

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

IKsPropertySet methods	Description
Set	Sets a property identified by a property set GUID and a property ID.
Get	Retrieves a property identified by a property set GUID and a property ID.
QuerySupported	Determines whether an object supports a property set.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IKsPropertySet::Get

IKsPropertySet Interface

Retrieves a property identified by a property set GUID and a property ID.

HRESULT Get(
  REFGUID *guidPropSet*,
  DWORD *dwPropID*,
  LPVOID *pInstanceData*,
  DWORD *cbInstanceData*,

LPVOID *pPropData*,
DWORD *cbPropData*,
DWORD **pcbReturned*
);

Parameters

guidPropSet
[in] Property set GUID.
dwPropID
[in] Identifier of the property within the property set.
pInstanceData
[out, size_is(cbInstanceData)] Pointer to instance data for the property.
cbInstanceData
[in] Number of bytes in the buffer to which *pInstanceData* points.
pPropData
[out, size_is(cbPropData)] Pointer to the retrieved buffer, which contains the value of the property.
cbPropData
[in] Number of bytes in the buffer to which *pPropData* points.
pcbReturned
[out] Pointer to the number of bytes returned in the buffer to which *pPropData* points.

Return Values

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

The current DirectShow implementation returns E_PROP_SET_UNSUPPORTED if the property set is not supported or E_PROP_ID_UNSUPPORTED if the property ID is not supported for the specified property set.

Remarks

To retrieve a property, allocate a buffer which this method will then fill in. To determine the necessary buffer size, specify NULL for *pPropData* and zero (0) for *cbPropData*. This method returns the necessary buffer size in *pcbReturned*.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IKsPropertySet::QuerySupported

IKsPropertySet Interface

Determines whether an object supports a property set.

HRESULT QuerySupported(
REFGUID *guidPropSet*,
DWORD *dwPropID*,
DWORD **pTypeSupport*
);

Parameters

guidPropSet
[in] Property set GUID.
dwPropID
[in] Identifier of the property within the property set.
pTypeSupport
[out] Pointer to a value in which to store flags indicating the support provided by the driver. Supported flags include the following:

Value	Meaning
KSPROPERTY_SUPPORT_GET	You can retrieve the property by calling the IKsPropertySet::Get method.
KSPROPERTY_SUPPORT_SET	You can change the property by calling IKsPropertySet::Set.

Return Values

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

The return values for the current DirectShow implementation include the following:

Value	Meaning
E_NOTIMPL	Property set is not supported.
E_PROP_ID_UNSUPPORTED	Property ID is not supported for the specified property set.
E_PROP_SET_UNSUPPORTED	Property set is not supported.
S_OK	Specified property set and property ID combination is supported.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IKsPropertySet::Set

IKsPropertySet Interface

Sets a property identified by a property set GUID and a property ID.

HRESULT Set(

REFGUID *guidPropSet*,
DWORD *dwPropID*,
LPVOID *pInstanceData*,
DWORD *cbInstanceData*,
LPVOID *pPropData*,
DWORD *cbPropData*
);

Parameters

guidPropSet
[in] Property set GUID.
dwPropID
[in] Identifier of the property within the property set.
pInstanceData
[out, size_is(cbInstanceData)] Pointer to instance data for the property.
cbInstanceData
[in] Number of bytes in the buffer to which *pInstanceData* points.
pPropData
[out, size_is(cbPropData)] Pointer to the retrieved buffer, which contains the value of the property.
cbPropData
[in] Number of bytes in the buffer to which *pPropData* points.

Return Values

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

The current DirectShow implementation returns E_PROP_SET_UNSUPPORTED if the property set is not supported or E_PROP_ID_UNSUPPORTED if the property ID is not supported for the specified property set.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaControl Interface

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

When to Implement

This interface is implemented by the filter graph manager. Implement this only if you are writing a plug-in distributor that needs to export the control methods. The CMediaControl base

class implements this interface and handles the IDispatch interface.

When to Use

Use this interface from any application that wants to control the playing of media through Microsoft® DirectShow™ filter graphs. Applications can also use it to enumerate the filters in the filter graph and all the filters in the registry, to add a source filter to the filter graph, and to instruct the filter graph manager to build a filter graph capable of rendering the media type in a file.

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.

IMediaControl methods	Description
Run	Switches the entire filter graph into running mode.
Pause	Pauses all filters in the filter graph.
Stop	Switches all filters in the filter graph to a stopped state.
StopWhenReady	Waits for an operation such as Pause to complete, allowing filters to queue up data, then stops the filter graph.
GetState	Retrieves the state of the filter graph.
RenderFile	Adds and connects filters needed to play the specified file.
AddSourceFilter	Adds to the graph the source filter that can read the given file name, and returns an IDispatch interface pointer representing the filter object.
get_FilterCollection	Retrieves a collection of IFilterInfo interfaces representing the filters in the graph.
get_RegFilterCollection	Retrieves a collection of IRegFilterInfo interfaces representing the filters available in the registry.

IMediaControl::AddSourceFilter

IMediaControl Interface

Adds to the graph the source filter that can read the given file name, and returns an IDispatch interface pointer representing the filter.

HRESULT AddSourceFilter(
BSTR *strFilename*,
IDispatch ***ppUnk*
);

Parameters

strFilename
[in] Name of the file containing the source video.
ppUnk
[out] Pointer to the IFilterInfo interface on the filter.

Return Values

Returns an HRESULT value.

Remarks

This method is primarily for use by Automation clients because it returns an IDispatch interface pointer. C and C++ applications should call the IGraphBuilder::AddSourceFilter method to perform this operation for maximum efficiency.

Previous Home Topic Contents Index Next

IMediaControl::get_FilterCollection

IMediaControl Interface

Retrieves a collection of IFilterInfo interfaces representing the filters in the graph and returns IDispatch for an object that supports the IAMCollection interface.

HRESULT get_FilterCollection(
IDispatch ***ppUnk*
);

Parameters

ppUnk
[out, retval] The IAMCollection interface on a collection of IFilterInfo objects.

Return Values

Returns an HRESULT value.

Remarks

This method is primarily for use by Automation clients because it returns an IDispatch interface pointer. C and C++ applications should call the IFilterGraph::EnumFilters method to perform this operation for maximum efficiency.

Previous Home Topic Contents Index Next

IMediaControl::get_RegFilterCollection

IMediaControl Interface

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

HRESULT get_RegFilterCollection(
IDispatch ***ppUnk*
);

Parameters

ppUnk
[out, retval] IDispatch interface of the IAMCollection object.

Return Values

Returns an HRESULT value.

Remarks

This method is primarily for use by Automation clients because it returns an IDispatch interface pointer. C and C++ applications should call the IFilterMapper::EnumMatchingFilters method to perform this operation for maximum efficiency.

Previous Home Topic Contents Index Next

IMediaControl::GetState

IMediaControl Interface

Retrieves the state of the filter graph.

```
HRESULT GetState(
  LONG msTimeout,
  OAFilterState* pfs
  );
```

Parameters

msTimeout
[in] Duration of the time-out, in milliseconds.
pfs
[out] Holds the returned state of the filter graph.

Return Values

Returns VFW_S_STATE_INTERMEDIATE if the state transition is not complete, or S_OK if it completed successfully.

Remarks

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

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

This method will return an error if there is a call on another thread to change the state while this method is blocked.

Avoid specifying a time-out of INFINITE. Threads cannot process messages while waiting in **GetState**. If you call **GetState** from the thread that processes Windows® messages, specify only small wait times on the call in order to remain responsive to user input. This is most important when streaming data from a source such as the Internet, because state transitions can take significantly more time to complete.

If you want to pause a filter graph completely before stopping it, call IMediaControl::Pause, and then IMediaControl::StopWhenReady (instead of calling **GetState** with an INFINITE time-out, and then IMediaControl::Stop).

Although *pfs* is declared as a pointer to an OAFilterState value in **IMediaControl::GetState**, DirectShow implements it as a pointer to a FILTER_STATE value in CBaseFilter::GetState and its derivatives. Since both OAFilterState and **FILTER_STATE** resolve to LONG values, this does not cause an error.

Previous Home Topic Contents Index Next

IMediaControl::Pause

IMediaControl Interface

Pauses all the filters in the filter graph.

HRESULT Pause( );

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IMediaControl::RenderFile

IMediaControl Interface

Adds and connects filters needed to play the specified file.

HRESULT RenderFile(
  BSTR *strFilename*
  );

Parameters

strFilename
Name of the file to render.

Return Values

Returns an HRESULT value.

Remarks

This method allows an application to pass the name of a media file that it wants rendered to the filter graph manager. The filter graph manager will build a graph of the filters needed to play back this file. This method is Automation-compatible and is equivalent to IGraphBuilder::RenderFile, which should be used by C and C++ applications.

Previous Home Topic Contents Index Next

IMediaControl::Run

IMediaControl Interface

Switches the entire filter graph into a running state.

HRESULT Run( );

Return Values

Returns an HRESULT value.

Remarks

In a running state, data is pushed down the filter graph and rendered. The graph remains in a running state until it is stopped by the IMediaControl::Pause or IMediaControl::Stop method. The graph remains in a running state even after notifying the application of completion (that is, the EC_COMPLETE notification is sent to the application). This allows the application to determine whether to pause or stop after completion.

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

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

Previous Home Topic Contents Index Next

IMediaControl::Stop

IMediaControl Interface

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

HRESULT Stop( );

Return Values

Returns an HRESULT value.

Remarks

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

Previous Home Topic Contents Index Next

IMediaControl::StopWhenReady

IMediaControl Interface

Waits for an operation such as Pause to complete, allowing filters to queue up data, then stops the filter graph.

HRESULT StopWhenReady( );

Return Values

Returns an HRESULT value.

Remarks

Changing the current position when stopped will not normally repaint the video window with the new position. Applications will need to enter Pause mode to do this. Calling **StopWhenReady** instead of simply calling Stop after this pause ensures that the graph is fully

paused, and that data has arrived at the video renderer and has been displayed before the graph is stopped.

This method is run asynchronously so that the application regains control immediately and can respond to user input. Use this method rather than calling IMediaControl::GetState with an INFINITE time-out, followed by IMediaControl::Stop.

Previous Home Topic Contents Index Next

IMediaEvent Interface

This interface supports event notification from the filter graph and filters within it to the application. It is decoupled by using a queuing scheme rather than callbacks, because events can be notified from worker threads that cannot safely call back into application code.

An event code and two DWORD values represent event notification information. Your application can use this for typical completion of asynchronous operations, errors that occur during asynchronous operation, or user-initiated events, such as when a user clicks a hot spot.

Filters within the filter graph and the filter graph itself raise event notifications. Possible events include playback completion or asynchronous playback errors. In addition, the filter graph provides a method to generate events at specific reference clock times. The filter graph exposes an IMediaEventSink interface that the filters within the graph can call to pass event notifications to the application.

Event notifications are placed in a queue. An application calls the IMediaEvent::GetEvent method to retrieve the next notification from the queue. This method blocks until there is an event to return. The GetEvent time-out parameter (*msTimeout*) allows the application to specify the time, in milliseconds, to wait for an event, including values of zero and INFINITE. After calling **GetEvent**, applications should always call FreeEventParams to release any resource associated with the event.

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

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

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

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

A seekable renderer is one that supports the IMediaPosition interface from the filter and that has only input pins, or whose input pins report through IPin::QueryInternalConnections that they are rendered. The filter graph uses **IPin::QueryInternalConnections** and **IMediaPosition** to detect seekable renderers. A seekable renderer should report EC_COMPLETE when all seekable streams on that filter have reached the stream's end.

A renderer can produce EC_COMPLETE (and a regular filter produce EndOfStream) for one of four reasons as follows:

- The typical case: Whether there is data arriving or not if it succeeds all calls to Receive it will eventually get EndOfStream. If the end of the media is reached and when all data and **EndOfStream** has been processed it will signal EC_COMPLETE.
- The filter can never produce any data. In that case it just passes EC_COMPLETE immediately when a Run method is called. For example, a filter would pass **EC_COMPLETE** if none if its input pins is connected.
- The complicated case, sometimes used by the wave renderer: It can't render data right now even though it's getting it but it may be able to later. In that case it fails the first Receive, schedules an EC_COMPLETE for a time *tStop* minus *tStart* in the future (based on the NewSegment parameters). If it finds it can start sending data it signals an EC_NEED_RESTART. A better approach could be to use stream control for this to avoid stopping and starting the graph.
- It detects an unrecoverable error. Then, like any filter, it signals end of stream which for a renderer means signaling EC_COMPLETE.

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

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

For a list of system-defined event notifications, see Event Notification Codes.

Note All events must be handled if a handle to an **IMediaEvent** interface is obtained

otherwise events will pile up and cause the heap to be used up.

When to Implement

The filter graph manager implements this interface.

You can use the CMediaEvent class, which handles the IDispatch implementation for Automation, to help implement this interface.

When to Use

Applications use this interface to retrieve event notifications or event handles from the filter graph manager. For example, an application can retrieve the EC_COMPLETE notification to find out when a media stream has been rendered completely.

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.

IMediaEvent methods	Description
GetEventHandle	Retrieves a handle to a manual-reset event that will be signaled.
GetEvent	Retrieves the next notification event.
WaitForCompletion	Blocks execution of the application thread until the graph's operation finishes.
CancelDefaultHandling	Cancels any default handling of the specified event by the filter graph.
RestoreDefaultHandling	Restores default handling for this event.
FreeEventParams	Frees resources associated with the parameters to an event.

Previous Home Topic Contents Index Next

IMediaEvent::CancelDefaultHandling

IMediaEvent Interface

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

HRESULT CancelDefaultHandling(
long *lEvCode*
);

Parameters

lEvCode
Event code for which to cancel default handling.

Return Values

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

Remarks

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

Previous Home Topic Contents Index Next

IMediaEvent::FreeEventParams

IMediaEvent Interface

Frees resources associated with the parameters of an event.

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

Parameters

lEventCode
[in] Next event notification.

lParam1
[in] First parameter of the event.
lParam2
[in] Second parameter of the event.

Return Values

Returns an HRESULT value.

Remarks

Event parameters can be of type LONG or BSTR. If a **BSTR** is passed as an event, it will have been allocated by the task allocator and should be freed using this method. No reference-counted interfaces are passed to an application using IMediaEvent::GetEvent because these cannot be overridden by IMediaEvent::CancelDefaultHandling. Therefore, this method is not used to release interfaces.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaEvent::GetEvent

IMediaEvent Interface

Retrieves the next notification event.

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

Parameters

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

Return Values

Returns an HRESULT value that depends on the implementation of the interface. If the time-out is zero and no event is waiting, or if the time-out elapses before an event appears, this method returns E_ABORT.

Remarks

The application can pass a time-out value of INFINITE to indicate that the method should block until there is an event; however, applications should avoid using INFINITE. Threads cannot process any messages while waiting in **GetEvent**. If you call **GetEvent** from the thread that processes Windows messages, specify only small wait times on the call in order to remain responsive to user input. This is most important when streaming data from a source such as the Internet, because state transitions can take significantly more time to complete.

After calling **GetEvent**, applications should always call FreeEventParams to release any resource associated with the event.

For a list of notification codes and event parameter values, see Event Notification Codes.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaEvent::GetEventHandle

IMediaEvent Interface

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

HRESULT GetEventHandle(
OAEVENT * *hEvent*
);

Parameters

hEvent
[out] Handle for the event.

Return Values

Returns an HRESULT value.

Remarks

You can monitor events (including the retrieved event) and messages on a single thread; to do this, declare a HANDLE variable, cast it to an OAEVENT pointer, then pass it to GetEventHandle. You must cast the pointer to an OAEVENT pointer because HANDLE is not a valid OLE Automation type. The following code fragment demonstrates how to cast and use the HANDLE.

```
HANDLE hEvent;
GetEventHandle( (OAEVENT*) &hEvent );
```

You can pass the retrieved event handle to the Microsoft Win32 WaitForMultipleObjects or MsgWaitForMultipleObjects functions to wait for event notifications at the same time as other messages and events on a single thread. The implementation of GetEvent sets and resets the handle within the application, so applications should not reset the handle themselves.

Previous Home Topic Contents Index Next

IMediaEvent::RestoreDefaultHandling

IMediaEvent Interface

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

HRESULT RestoreDefaultHandling(
long *lEvCode*
);

Parameters

lEvCode
[in] Event to restore.

Return Values

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

Remarks

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

Previous Home Topic Contents Index Next

IMediaEvent::WaitForCompletion

IMediaEvent Interface

Blocks execution of the application thread until the graph's operation finishes.

```
HRESULT WaitForCompletion(
  long msTimeout,
  long * pEvCode
  );
```

Parameters

msTimeout
[in] Duration of the time-out, in milliseconds. Pass zero to return immediately. To block indefinitely, pass INFINITE.

pEvCode
[out] Event that terminated the wait. This value can be one of the following:

Value	Meaning
EC_COMPLETE	Operation completed.
EC_ERRORABORT	Error. Playback can't continue.
EC_USERABORT	User terminated the operation.
Zero (0)	Operation has not completed.

Return Values

Returns one of the following HRESULT values.

Value	Meaning
E_ABORT	Function timed out before the operation completed. This is equivalent to a zero *pEvCode* value.
S_OK	Operation completed.

Remarks

This method is the equivalent of blocking until the event notification EC_COMPLETE, EC_ERRORABORT, or EC_USERABORT is received, or the time-out occurs.

When this method returns, the filter graph is still running. This method assumes that separate calls to the IMediaEvent interface are not being made. This method fails if the graph is not in or transitioning into a running state.

The time-out parameter (*msTimeout*) specifies the length of time to wait for completion. To test if the operation completed, specify a zero *msTimeout* value and check the event code value (*pEvCode*) for zero, indicating that the operation is not completed.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaEventEx Interface

This interface derives from IMediaEvent and adds a method that allows registration of a window to receive messages when events occur.

When to Implement

This interface is implemented by the filter graph manager.

Unlike IMediaEvent, IMediaEventEx is not available through Automation, and therefore cannot be called directly from Visual Basic.

The CMediaEvent Class supports IMediaEventEx.

When to Use

This interface is used by applications to receive notification that an event has occurred. Applications can then avoid using a separate thread that waits until an event is set.

For a list of system-defined event notifications, see DirectShow Event Notification Codes.

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.

IMediaEvent methods	Description
GetEventHandle	Retrieves a handle to a manual-reset event that will be signaled.
GetEvent	Retrieves the next notification event.
WaitForCompletion	Waits until the graph's operation has completed.
CancelDefaultHandling	Cancels any default handling of the specified event by the filter graph.
RestoreDefaultHandling	Restores default handling for this event.
FreeEventParams	Frees resources associated with the parameters to an event.

IMediaEventEx methods	Description
SetNotifyWindow	Registers a window that will handle messages when a specified event occurs.
SetNotifyFlags	Turns event notifications on or off.
GetNotifyFlags	Retrieves whether event notifications are on or off.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaEventEx::GetNotifyFlags

IMediaEventEx Interface

Retrieves whether event notifications are on or off.

HRESULT GetNotifyFlags(
[out] long **lplNoNotifyFlags*
);

Parameters

lplNoNotifyFlags
[out] Pointer to a value indicating whether event notifications should be on or off. 0x00 indicates notifications are on and 0x01 indicates notifications are off.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns S_OK if successful or E_POINTER if *lplNoNotifyFlags* is NULL.

Remarks

The handle returned by the GetEventHandle method will be signaled at end of stream.

Previous Home Topic Contents Index Next

IMediaEventEx::SetNotifyFlags

IMediaEventEx Interface

Turns event notifications on or off.

HRESULT SetNotifyFlags(
long *lNoNotifyFlags*
);

Parameters

lNoNotifyFlags
[in] Value indicating whether event notifications should be on or off. Specify 0x00 to turn notifications on or specify 0x01 to turn notification off.

Return Values

Returns an HRESULT value that depends on the implementation of the interface. The current DirectShow implementation returns S_OK if successful or E_INVALIDARG if the argument is invalid.

Remarks

If notification has been turned off, the handle returned by the GetEventHandle method will be signaled at end of stream.

Previous Home Topic Contents Index Next

IMediaEventEx::SetNotifyWindow

IMediaEventEx Interface

Registers a window that will handle messages in response to all events from an object.

HRESULT SetNotifyWindow(
OAHWND *hwnd*,
long *lMsg*,
long *lInstanceData*
);

Parameters

hwnd
[in] Handle of window to notify. Pass NULL to stop notification.
lMsg
[in] The window message to be passed as the notification.
lInstanceData
[in] Value (instance data) to be passed as the *lParam* parameter for the *lMsg* message.

Return Values

Returns S_OK if successful or E_INVALIDARG if an argument is invalid.

Remarks

This method designates a window as the recipient of messages generated by or sent to the current DirectShow object. You can use this method to monitor the messages from multiple sources in a single window, which lowers processing overhead.

If an event occurs, DirectShow posts the notification message specified in *lMsg* to the window specified by *hwnd*. **SetNotifyWindow** sets the message's *lParam* parameter to *lInstanceData* and its *wParam* parameter to zero.

You can retrieve the event information by calling the GetEvent method when the destination window receives the message.

All event types post the message; when it arrives, any number of events, including zero, might actually be in the queue. If more than one event is in the queue when you call **SetNotifyWindow**, DirectShow posts only one message. If the application can receive the message after some other action has cleared events from the queue, there might be no events in the queue. For example, the Stop method clears all EC_COMPLETE events from the queue.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

IMediaEventSink Interface

The filter graph manager exposes the **IMediaEventSink** interface; it is called from filters to