Name	Description
BreakConnect	Informs the derived class when the connection is broken.
CheckConnect	Informs the derived class when the connection process is starting.
CheckMediaType	Determines if the pin can use a specified media type.
CheckStreaming	Verifies conditions for continuing with a streaming operation.
CompleteConnect	Informs the derived class when the connection process has completed.
SetMediaType	Informs the derived class when the media type is established for the connection.

Implemented IPin Methods

Name	Description
BeginFlush	Informs the pin to begin a flush operation.
EndFlush	Informs the pin to end a flush operation and notifies the pin that it can start accepting data again.
EndOfStream	Informs the input pin that no additional data is expected until a new run command is issued.
NewSegment	Specifies that samples following this call are grouped as a segment with a given start time, stop time, and rate.
QueryId	Retrieves an identifier for the pin.

Implemented IMemInputPin Methods

Name	Description
Receive	Receives the next block of data from the stream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformInputPin::BeginFlush

CTransformInputPin Class

Informs the pin to begin a flush operation.

HRESULT BeginFlush(void);

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::BeginFlush method and overrides the CBaseInputPin::BeginFlush member function. It checks to see if the pin is connected, and then calls **CBaseInputPin::BeginFlush**, and finally calls the CTransformFilter::BeginFlush member function.

Previous Home Topic Contents Index Next

CTransformInputPin::BreakConnect

CTransformInputPin Class

Informs the derived class when the connection is broken.

HRESULT BreakConnect( );

Return Values

Returns NOERROR in this implementation.

Remarks

This member function overrides the CBasePin::BreakConnect member function and calls the CTransformFilter::BreakConnect member function. Override **CTransformFilter::BreakConnect** to undo anything carried out in CTransformInputPin::CheckConnect (such as releasing extra interfaces).

Previous Home Topic Contents Index Next

CTransformInputPin::CheckConnect

CTransformInputPin Class

Informs the derived class when the connection process is starting.

HRESULT CheckConnect(
IPin **pPin*
);

Parameters

pPin
Pointer to the IPin interface of the connecting pin.

Return Values

Returns NOERROR by default.

Remarks

This member function overrides the CBasePin::CheckConnect member function and calls the CTransformFilter::CheckConnect member function. Override **CTransformFilter::CheckConnect** to add additional interfaces.

Previous Home Topic Contents Index Next

CTransformInputPin::CheckMediaType

CTransformInputPin Class

Determines if the pin can use a specified media type.

```
HRESULT CheckMediaType(
  const CMediaType* mtIn
  );
```

Parameters

mtIn
Pointer to a media type object.

Return Values

No return value.

Remarks

This member function calls the pure-virtual CTransformFilter::CheckInputType member function, which must be overridden when deriving a class from the CTransformFilter class. The overridden CheckInputType member function is responsible for determining which media types the input pin supports.

Previous Home Topic Contents Index Next

CTransformInputPin::CheckStreaming

CTransformInputPin Class

Verifies conditions for continuing with a streaming operation.

HRESULT CheckStreaming( );

Return Values

Returns one of the following HRESULT values, depending on the state.

Value	Meaning
S_FALSE	Currently in flushing state.
S_OK	Receive or EndOfStream operations can safely proceed.
VFW_E_NOT_CONNECTED	The output pin either does not exist or isn't connected.
VFW_E_RUNTIME_ERROR	A run-time error occurred when processing a previous sample.
VFW_E_WRONG_STATE	The filter is in the State_Stopped state.

Remarks

This member function overrides the CBaseInputPin::CheckStreaming member function and calls that base class implementation for most of the condition checks. It determines if the pin is connected, if it is in a paused or running state, and if it is not currently flushing data or processing a run-time error.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformInputPin::CompleteConnect

CTransformInputPin Class

Informs the derived class when the connection process has been completed.

HRESULT CompleteConnect(

IPin **pReceivePin*
);

Parameters

pReceivePin
Pointer to the input pin being connected to.

Return Values

Returns an HRESULT value.

Remarks

This member function overrides the CBasePin::CompleteConnect member function. It calls the base class **CBasePin::CompleteConnect** member function and then calls CTransformFilter::CompleteConnect.

Previous Home Topic Contents Index Next

CTransformInputPin::CTransformInputPin

CTransformInputPin Class

Constructs a CTransformInputPin object.

CTransformInputPin(
TCHAR **pObjectName*,
CTransformFilter **pTransformFilter*,
HRESULT * *phr*,
LPCWSTR *pName*
);

Parameters

pObjectName
Name of the CTransformInputPin object.
pTransformFilter
Pointer to the CTransformFilter class.
phr
Pointer to an HRESULT value in which to return resulting information. This should be modified only if a failure occurs. If it is a failure code on input, construction can be terminated, but in any case the destructor will be called by the creator when the **HRESULT** error is detected.
pName
Name of the pin.

Return Values

No return value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformInputPin::CurrentMediaType

CTransformInputPin Class

Retrieves the media type currently assigned to the filter.

CMediaType& CurrentMediaType( );

Return Values

Returns the value of CBasePin::m_mt.

Previous Home Topic Contents Index Next

CTransformInputPin::EndFlush

CTransformInputPin Class

Informs the pin to end a flush operation and notifies the pin that it can start accepting data again.

HRESULT EndFlush(void);

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::EndFlush method and overrides the CBaseInputPin::EndFlush member function. It checks to see if the pin is connected, calls the **CBaseInputPin::EndFlush** member function, and finally calls the CTransformFilter::EndFlush member function.

Previous Home Topic Contents Index Next

CTransformInputPin::EndOfStream

CTransformInputPin Class

Informs the input pin that no additional data is expected until a new run command is issued.

HRESULT EndOfStream(void);

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::EndOfStream method. It calls CTransformInputPin::CheckStreaming to see that the filter is in a streaming state and then calls the CTransformFilter::EndOfStream member function.

Previous Home Topic Contents Index Next

CTransformInputPin::NewSegment

CTransformInputPin Class

Specifies that samples following this call are grouped as a segment with a given start time, stop time, and rate.

HRESULT NewSegment(
REFERENCE_TIME *tStart*,
REFERENCE_TIME *tStop*,
double *dRate*
);

Parameters

tStart
Start time of the segment.
tStop
Stop time of the segment.
dRate
Rate of the segment.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::NewSegment method and overrides the CBasePin::NewSegment member function. It calls the base class implementation first (**CBasePin::NewSegment**), and then calls CTransformFilter::NewSegment to pass the notification on to the next filter downstream.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformInputPin::QueryId

CTransformInputPin Class

Retrieves an identifier for the pin.

```
HRESULT QueryId(
  LPWSTR * Id
  );
```

Parameters

Id
Pin identifier.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::QueryId method and overrides the CBasePin::QueryId member function. It returns the name "In". The caller is responsible for freeing the memory by using the Microsoft® Win32® CoTaskMemFree function.

Previous Home Topic Contents Index Next

CTransformInputPin::Receive

CTransformInputPin Class

Receives the next block of data from the stream.

HRESULT Receive(
IMediaSample * *pSample*
);

Parameters

pSample
Pointer to a media sample.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IMemInputPin::Receive method. Add a reference to the block of data if access to it is required after the completion of this method. For instance, some decoder filters for temporal compression data streams require that the previous sample be kept in order to decode the current sample.

This member function calls the CTransformFilter::Receive member function, which does the work of calling the transform function and then passing the sample on.

Previous Home Topic Contents Index Next

CTransformInputPin::SetMediaType

CTransformInputPin Class

Informs the derived class when the media type is established for the connection.

HRESULT SetMediaType(
const CMediaType* *mt*
);

Parameters

mt
Pointer to an input media type to be used.

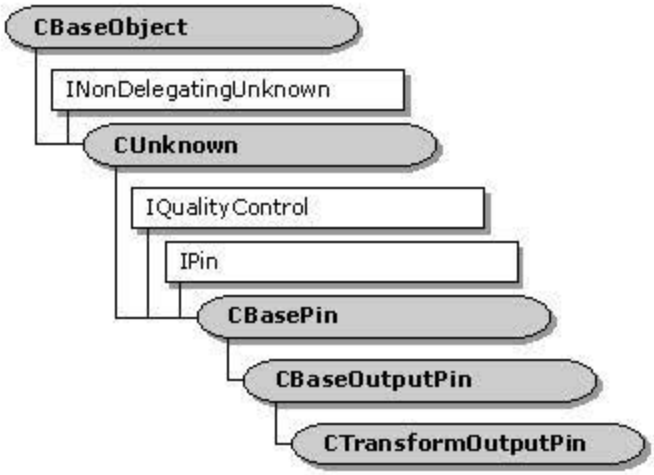
Return Values

Returns an HRESULT value.

Remarks

This member function overrides the CBasePin::SetMediaType member function. It calls the base class **CBasePin::SetMediaType** member function, which returns NOERROR, and then calls CTransformFilter::SetMediaType, which the derived class can override to be informed when the media type is set.

CTransformOutputPin Class



The **CTransformOutputPin** class implements the output pin of a simple transform filter. It is the class assigned to the m_pOutput data member of the CTransformFilter class. Typically, you can create objects of a class derived from **CTransformFilter** without modifying the **CTransformOutputPin** class. If you want to override this class and derive a class from **CTransformFilter**, use the class and then override the CTransformFilter::GetPin member function to create pins of your derived class.

Protected Data Members

Name	Description
m_pTransformFilter	Pointer to the owning CTransformFilter object.

Public Data Members

Name	Description
m_pPosition	Pointer to a CPosPassThru object that implements the IMediaPosition interface to pass media position commands on to the upstream filter.

Member Functions

Name	Description
CTransformOutputPin	Constructs a CTransformOutputPin object.
CurrentMediaType	Retrieves the media type currently assigned to the filter.

Overridable Member Functions

Name	Description
BreakConnect	Informs the derived class when the connection is broken.
CheckConnect	Informs the derived class when the connection process is starting.
CheckMediaType	Determines if the pin can use a specified media type.
CompleteConnect	Informs the derived class when the connection process has completed.
DecideBufferSize	Determines the number and size of buffers required.
GetMediaType	Returns the media type that the output pin uses.
SetMediaType	Informs the derived class when the media type is established for the connection.

Implemented IQualityControl Methods

Name	Description
Notify	Receives a quality-control notification, typically from a downstream filter. This method is inherited from the IQualityControl interface through the CBasePin class.

Implemented IPin Methods

Name	Description
QueryId	Retrieves an identifier for the pin.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingQueryInterface	Returns an interface and increments the reference count.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformOutputPin::BreakConnect

CTransformOutputPin Class

Informs the derived class when the connection is broken.

HRESULT BreakConnect( );

Return Values

Returns NOERROR.

Remarks

This member function overrides the CBaseOutputPin::BreakConnect member function and calls

the CTransformFilter::BreakConnect member function. It then calls the base class implementation in **CBaseOutputPin::BreakConnect**. Override **CTransformFilter::BreakConnect** to undo anything carried out in the CTransformOutputPin::CheckConnect member function (for example, releasing interfaces previously added to the reference count).

Previous Home Topic Contents Index Next

CTransformOutputPin::CheckConnect

CTransformOutputPin Class

Informs the derived class when the connection process is starting.

HRESULT CheckConnect(
IPin **pPin*
);

Parameters

pPin
Pointer to the IPin interface of the connecting pin.

Return Values

Returns NOERROR by default.

Remarks

This member function overrides the CBasePin::CheckConnect member function and calls the CTransformFilter::CheckConnect member function. It then calls the base class implementation in CBaseOutputPin::CheckConnect. Override **CTransformFilter::CheckConnect** to add additional interfaces.

Previous Home Topic Contents Index Next

CTransformOutputPin::CheckMediaType

CTransformOutputPin Class

Determines if the input pin supports a specified media type.

HRESULT CheckMediaType(
const CMediaType* *mtIn*
);

Parameters

mtIn
Pointer to a media type object.

Return Values

No return value.

Remarks

This member function calls the pure-virtual CTransformFilter::CheckTransform member function, which must be overridden when deriving a class from the CTransformFilter class. The overridden **CTransformFilter::CheckTransform** member function determines which media types the output pin supports.

Previous Home Topic Contents Index Next

CTransformOutputPin::CompleteConnect

CTransformOutputPin Class

Informs the derived class when the connection process has completed.

HRESULT CompleteConnect(
IPin **pReceivePin*
);

Parameters

pReceivePin
Pointer to the output pin that is being connected to.

Return Values

Returns an HRESULT value.

Remarks

This member function overrides the CBaseOutputPin::CompleteConnect member function and calls the CTransformFilter::CompleteConnect member function, which returns NOERROR by default. It then calls the base class implementation in **CBaseOutputPin::CompleteConnect**. Override the **CTransformFilter::CompleteConnect** member function to retrieve any additional interfaces not retrieved by the base class that your output pin might need from the connected pin.

Previous Home Topic Contents Index Next

CTransformOutputPin::CTransformOutputPin

CTransformOutputPin Class

Constructs a CTransformOutputPin object.

CTransformOutputPin(
TCHAR **pObjectName*,
CTransformFilter **pTransformFilter*,
HRESULT * *phr*,
LPCWSTR *pName*
);

Parameters

pObjectName
Name of the CTransformOutputPin object.
pTransformFilter
Pointer to the CTransformFilter class.
phr
Pointer to an HRESULT value in which to return resulting information. This should be modified only if a failure occurs. If it is a failure code on input, construction can be aborted, but in any case the destructor will be called by the creator when the **HRESULT** error is detected.
pName
Name of the pin.

Return Values

No return value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformOutputPin::CurrentMediaType

CTransformOutputPin Class

Retrieves the media type currently assigned to the filter.

CMediaType& CurrentMediaType( );

Return Values

Returns the value of CBasePin::m_mt.

Previous Home Topic Contents Index Next

CTransformOutputPin::DecideBufferSize

CTransformOutputPin Class

Determines the number and size of buffers required.

HRESULT DecideBufferSize(
IMemAllocator * *pAlloc*,
ALLOCATOR_PROPERTIES * *ppropInputRequest*
);

Parameters

pAlloc
Allocator assigned to the transfer.
ppropInputRequest
Requested allocator properties for count, size, and alignment, as specified by the ALLOCATOR_PROPERTIES structure.

Return Values

Returns an HRESULT value.

Remarks

This member function overrides the CBaseOutputPin::DecideBufferSize member function and calls the pure virtual CTransformFilter::DecideBufferSize member function, which your derived

class must override and implement. This member function is called from the CBaseOutputPin class during the connection process.

Previous Home Topic Contents Index Next

CTransformOutputPin::GetMediaType

CTransformOutputPin Class

Returns the media type for the output pin to use.

```
HRESULT GetMediaType(
  int iPosition,
  CMediaType *pMediaType
  );
```

Parameters

iPosition
Position of the media type in the media type list.
pMediaType
Returned media type object.

Return Values

Returns an HRESULT value that depends on the implementation of the pure virtual CTransformFilter::GetMediaType member function. **HRESULT** can include one of the following constants.

Value	Meaning
NOERROR	A media type is returned.
S_FALSE	Although the *iPosition* parameter typically is valid, it does not correspond to a media type that is currently valid.
VFW_S_NO_MORE_ITEMS	The *iPosition* parameter is beyond the valid range.

Use other standard error values, such as E_INVALIDARG, for error cases.

Remarks

This member function overrides the CBasePin::GetMediaType member function and calls the pure virtual CTransformFilter::GetMediaType member function, which must be overridden to return media types supported by your filter. This is part of the implementation of CBasePin::EnumMediaTypes.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransformOutputPin::NonDelegatingQueryInter

CTransformOutputPin Class

Returns an interface and increments the reference count.

HRESULT NonDelegatingQueryInterface(
REFIID *riid*,
void ** *ppv*
);

Parameters

riid
Reference identifier.
ppv
Pointer to the interface.

Return Values

Returns E_POINTER if *ppv* is invalid. Returns NOERROR if the query is successful. If the query is unsuccessful and the requested interface is IMediaPosition or IMediaSeeking, returns an HRESULT from a call to CreatePosPassThru. If the query is unsuccessful and the interface is not **IMediaPosition** or **IMediaSeeking**, returns E_NOINTERFACE.

Remarks

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method. It overrides the CBasePin::NonDelegatingQueryInterface member function and passes references to the IPin, IQualityControl, IMediaPosition, IMediaSeeking, and IUnknown interfaces. Override this class to return other interfaces on the object in the derived class.

Previous Home Topic Contents Index Next

CTransformOutputPin::Notify

CTransformOutputPin Class

Notifies the recipient that a quality change is requested.

HRESULT Notify(
IBaseFilter * *pSelf*,
Quality *q*
);

Parameters

pSelf
Pointer to the filter that is sending the quality notification.
q
Quality notification structure.

Return Values

Default base class implementation returns E_FAIL.

Remarks

This member function implements the IQualityControl::Notify method and overrides the CBasePin::Notify member function. It calls the CTransformFilter::AlterQuality member function to determine if the filter can do something to adjust the quality of the media stream (such as discarding samples). If that member function returns S_FALSE, it calls the CBaseInputPin::PassNotify member function, which passes the notification to the upstream filter after verifying that it is connected upstream.

Previous Home Topic Contents Index Next

CTransformOutputPin::QueryId

CTransformOutputPin Class

Retrieves an identifier for the pin.

HRESULT QueryId(
LPWSTR * *Id*
);

Parameters

Id
Pin identifier.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IPin::QueryId method and overrides the CBasePin::QueryId member function. It returns the name "Out". The caller is responsible for freeing the memory by using the Microsoft® Win32® CoTaskMemFree function.

Previous Home Topic Contents Index Next

CTransformOutputPin::SetMediaType

CTransformOutputPin Class

Sets the media type for the connection to use.

```
HRESULT SetMediaType(
  const CMediaType* mt
  );
```

Parameters

mt
Pointer to an output media type to be used.

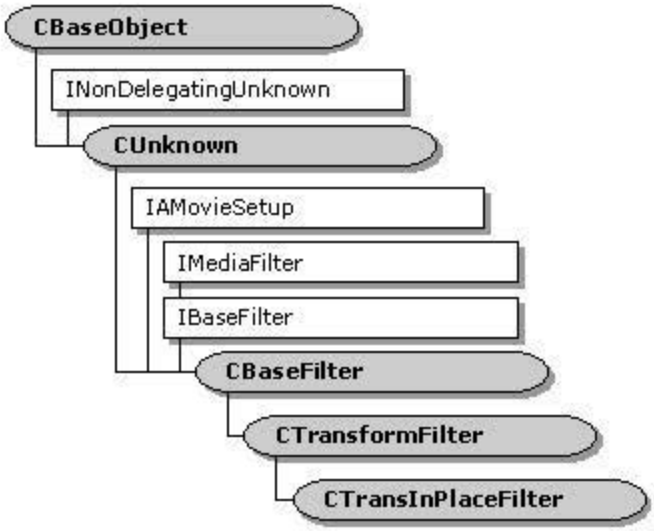
Return Values

Returns an HRESULT value (NOERROR by default).

Remarks

This member function overrides the CBasePin::SetMediaType member function and calls the CTransformFilter::SetMediaType member function with the direction set to output. Override **CTransformFilter::SetMediaType** to handle any conditions that you want handled at this time in the connection process.

CTransInPlaceFilter Class



CTransInPlaceFilter is an abstract base class that provides support for a simple transform filter with a single input and a single output. It is derived from the CUnknown class, and supports the IBaseFilter interface, the IMediaFilter interface, and two pins. Each pin supports the IPin interface and uses the shared memory transport based on the IMemInputPin interface. The filter uses classes derived from the CBaseMediaFilter class to support **IBaseFilter** and **IMediaFilter**. The input pin is derived from the CBaseInputPin class, and the output pin is derived from the CBaseOutputPin class.

For more information about using this class to create a transform filter, see Creating a Transform Filter.

Protected Data Members

Name	Description
m_idTransInPlace	Performance-measuring identifier.

Member Functions

Name	Description
Copy	Returns a pointer to an identical copy of a media sample.
CTransInPlaceFilter	Constructs a CTransInPlaceFilter object.
InputPin	Returns a pointer to the input pin associated with the filter.
OutputPin	Returns a pointer to the output pin associated with the filter.

Overridable Member Functions

Name	Description
CheckTransform	Verifies that the media type is supported by input and output pins.
CompleteConnect	Reconnects the input or output pin if necessary.
DecideBufferSize	Determines the size of the transport buffer.
GetMediaType	Returns the media type to be used by the output pin.
GetPin	Returns a pin if an index is specified.
Receive	Receives the sample, calls the derived class's Transform member function, and then delivers the sample.
RegisterPerfId	Registers a performance measurement identifier (if PERF is defined).
Transform	Performs transformation operations in place on the IMediaSample interface (pure virtual).

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceFilter::CheckTransform

CTransInPlaceFilter Class

Verifies that the media is supported by input and output pins.

```
HRESULT CheckTransform(
  const CMediaType *mtIn,
  const CMediaType *mtOut
  );
```

Parameters

mtIn
Input pin media type.
mtOut
Output pin media type.

Return Values

Returns S_OK by default.

Remarks

This member function overrides the CTransformFilter::CheckTransform member function. The base class functions that call this member function are overridden in this class to call the CTransformFilter::CheckInputType member function that is overridden in the derived class, with the assumption that the type does not change. Usually there is no reason for this member

function to be called. In debug builds some calls will be made, and returning S_OK ensures that these calls do not assert.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::CompleteConnect

CTransInPlaceFilter Class

Reconnects the input or output pin if necessary.

HRESULT CompleteConnect(
PIN_DIRECTION *direction*,
IPin **pReceivePin*
);

Parameters

direction
Pin direction.
pReceivePin
Pointer to the output pin to which to connect.

Return Values

Returns NOERROR if successful; otherwise, returns VFW_E_NOT_IN_GRAPH if the filter is not part of a graph, or returns an HRESULT that indicates the error. **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.

Remarks

This member function overrides the CTransformFilter::CompleteConnect member function. It is called by one of the pin classes at the end of a successful connection. Because the input and output pins must both use the same allocator, this member function reconnects the opposite pin if necessary.

When the input pin is first connected, the output pin has not yet been connected and the downstream filter's allocator is unknown, so the allocator for the input pin is chosen to be the upstream pin's allocator. When the transform filter's output pin is connected, however, it has access to the downstream filter's allocator and should force a reconnect on the input pin and

offer that allocator. When the input pin is reconnected, it forces a reconnect on the output pin if the allocator chosen for the input pin's connection differs from the output pin's connection. This member function supplies the reconnection for either output or input pins.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceFilter::Copy

CTransInPlaceFilter Class

Creates a copy of the specified media sample.

IMediaSample * CTransInPlaceFilter::Copy(
IMediaSample **pSource*
);

Parameters

pSource
Pointer to an object that implements the IMediaSample interface.

Return Values

Returns a pointer to the new sample.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::CTransInPlaceFilter

CTransInPlaceFilter Class

Constructs a CTransInPlaceFilter object.

CTransInPlaceFilter(
TCHAR * *pObjectName*,
LPUNKNOWN *lpUnk*,
REFCLSID *clsid*,

HRESULT * *phr*
);

Parameters

pObjectName
Name given to the CTransInPlaceFilter object.
lpUnk
Pointer to LPUNKNOWN.
clsid
Class identifier of the CTransInPlaceFilter class.
phr
Pointer to the HRESULT value for resulting information.

Return Values

No return value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceFilter::DecideBufferSize

CTransInPlaceFilter Class

Determines the size of the transport buffer.

HRESULT DecideBufferSize(
IMemAllocator * *pAlloc*,
ALLOCATOR_PROPERTIES * *pProperties*
);

Parameters

pAlloc
Pointer to the IMemAllocator object used by the output pin.
pProperties
Requested allocator properties for count, size, and alignment, as specified by the ALLOCATOR_PROPERTIES structure.

Return Values

Returns NOERROR if successful; otherwise, returns an HRESULT value indicating the error. **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.

Remarks

This member function overrides the CTransformFilter::DecideBufferSize member function. It is called when the filter must provide its own allocator. Allocator requirements are obtained from the filter's input pin and passed to the output pin.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceFilter::GetMediaType

CTransInPlaceFilter Class

Returns the media type to be used by the output pin.

```
HRESULT GetMediaType(
  int iPosition,
  CMediaType *pMediaType
  );
```

Parameters

iPosition
Position of the media type in the media type list.

pMediaType
Returned media type object.

Return Values

Returns E_UNEXPECTED because it is not expected to be called.

Remarks

In the CTransformFilter class, this member function is called by the associated input or output pin class's GetMediaType member function to retrieve the next media type in the list and return it to the pin's CBasePin::EnumMediaTypes member function.

However, in the CTransInPlaceFilter class, the pin classes override the CBasePin::EnumMediaTypes member function so that it bypasses the filter and calls the enumerator of the opposite connected pin. (For example, the output pin enumerator uses the upstream filter's enumerator, and the input pin uses the connected downstream filter's enumerator.) Therefore, this member function should never be called by the inplace pin classes. It is implemented to prevent "undefined pure virtual" compiler warnings.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::GetPin

CTransInPlaceFilter Class

Returns a pin if an index is specified.

```
virtual CBasePin * GetPin(
  int n
  );
```

Parameters

n
Index of the pin to return.

Return Values

Returns a pointer to a CBasePin object.

Remarks

This member function is implemented and need not be overridden unless one or more of the transform pin classes (CTransInPlaceInputPin or CTransInPlaceOutputPin) are being overridden.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::InputPin

CTransInPlaceFilter Class

Retrieves a pointer to the input pin associated with the filter.

CTransInPlaceInputPin *InputPin( );

Return Values

Returns a pointer to a CTransInPlaceInputPin object.

Remarks

This member function is protected.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::OutputPin

CTransInPlaceFilter Class

Retrieves a pointer to the output pin associated with the filter.

CTransInPlaceOutputPin *OutputPin( );

Return Values

Returns a pointer to a CTransInPlaceOutputPin object.

Remarks

This member function is protected.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::Receive

CTransInPlaceFilter Class

Receives the media sample, calls the CTransInPlaceFilter::Transform member function, and then delivers the media sample.

HRESULT Receive(
IMediaSample **pSample*
);

Parameters

pSample
Sample to deliver.

Return Values

Returns an HRESULT value that depends on the implementation of the derived class' Transform function . **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 member function overrides the CTransformFilter::Receive member function. Override it only if you need more control of the process.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceFilter::RegisterPerfId

CTransInPlaceFilter Class

Registers a performance measurement identifier.

virtual void RegisterPerfId( );

Return Values

No return value.

Remarks

By default, this member function registers the performance identifier (m_idTransform) with the string "TransinPlace". Override this member function to register a performance measurement with a less generic string. This should be done to avoid confusion with other filters. This member function is enabled only when PERF is defined.

Previous Home Topic Contents Index Next

CTransInPlaceFilter::Transform

CTransInPlaceFilter Class

Transforms the data in *pSample* in place.

```
virtual HRESULT Transform(
  IMediaSample *pSample
  ) PURE;
```

Parameters

pSample
Pointer to the input IMediaSample interface.

Return Values

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

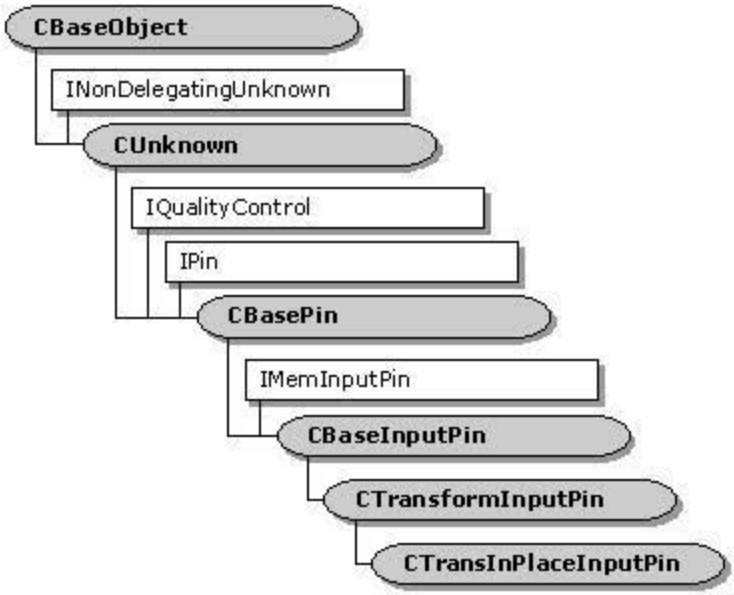
Value	Meaning
E_FAIL	Failure.
E_POINTER	Null pointer argument.
E_INVALIDARG	Invalid argument.
E_NOTIMPL	Method isn't supported.
S_OK or NOERROR	Success.

Remarks

You must supply this member function in the derived class to perform the actual work of your filter. This member function is called by CTransInPlaceFilter::Receive before passing the sample on to the downstream filter. **Transform** can return S_FALSE to indicate that the sample should not be delivered downstream.

Previous Home Topic Contents Index Next

CTransInPlaceInputPin Class



The **CTransInPlaceInputPin** class implements the input pin of a transform-inplace filter (CTransInPlaceFilter). This is part of a transform filter that transforms data in place rather than making a copy of it. The CTransInPlaceFilter::InputPin member function returns a pointer to **CTransInPlaceInputPin** object.

Typically, you can create objects of a class derived from **CTransInPlaceInputPin** without modifying this class. That is, you can usually override member functions in the CTransInPlaceFilter class that member functions of the **CTransInPlaceInputPin** class call, and not have to derive your own classes for either of the pin classes.

However, if you want to override this class and derive your filter class from CTransInPlaceFilter, you must override the CTransInPlaceFilter::GetPin member function to create pins of your derived class.

Protected Data Members

Name	Description
m_bReadOnly	Flag to indicate if the stream is read-only.
m_pTIPFilter	Pointer to the CTransInPlaceFilter object that owns this pin.

Member Functions

Name	Description
CTransInPlaceInputPin	Constructs a CTransInPlaceInputPin object.
PeekAllocator	Returns a pointer to the default allocator.
ReadOnly	Returns m_bReadOnly to indicate whether or not a stream is read-only.

Overridable Member Functions

Name	Description
CheckMediaType	Determines if the pin can use a specified media type.

Implemented IPin Methods

Name	Description
EnumMediaTypes	Provides a media type enumerator from the downstream filter.

Implemented IMemInputPin Methods

Name	Description
GetAllocator	Retrieves the upstream allocator.
GetAllocatorRequirements	Passes requests for allocator requirements downstream.
NotifyAllocator	Receives notification of which allocator the connected output pin will use.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::CheckMediaType

CTransInPlaceInputPin Class

Determines if the media type is acceptable.

```
HRESULT CheckMediaType(
  const CMediaType* pmt
  );
```

Parameters

pmt
Media type being checked.

Return Values

Returns an HRESULT value that depends on the implementation of the owning filter's CTransformFilter::CheckInputType member function. **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 member function overrides the CTransformInputPin::CheckMediaType member function. It first calls the owning filter's CheckInputType member function. (This is a purely virtual function which must be overridden when deriving a class from the CTransformFilter class. The overridden CheckInputType member function determines which media types the input pin supports.) Then, if the filter's output pin is not connected, this member function agrees to any media type. If the output pin is connected, it asks the downstream connected input pin if it accepts this type and returns the result.

The CheckInputType member function must be overridden by the class of the owning filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::CTransInPlaceInputPin

CTransInPlaceInputPin Class

Constructs a CTransInPlaceInputPin object.

CTransInPlaceInputPin(
TCHAR **pObjectName*,
CTransInPlaceFilter **pFilter*,
HRESULT * *phr*,
LPCWSTR *pName*
);

Parameters

pObjectName
Name of the CTransInPlaceInputPin class object.

pFilter
Pointer to the CTransInPlaceFilter class.

phr
Pointer to an HRESULT value in which to return resulting information. This should be modified only if a failure occurs. If it is a failure code on input, construction can be

terminated; but in any case the destructor will be called by the creator when the **HRESULT** error is detected.

pName

Name of the pin.

Return Values

No return value.

Remarks

This member function doesn't create the pins. The pins are created when they are first required. All external attempts to access pins (by enumeration or by CBaseFilter::FindPin) go through CTransInPlaceFilter::GetPin, which creates the pins initially.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::EnumMediaTypes

CTransInPlaceInputPin Class

Provides an enumerator for media types by retrieving one from downstream.

HRESULT EnumMediaTypes(
IEnumMediaTypes ***ppEnum**
);

Parameters

ppEnum

[out] Pointer to an enumerator for the media types.

Return Values

Returns NOERROR if successful, VFW_E_NOT_CONNECTED if there is no connection, or an HRESULT that indicates an error with the enumerator, such as E_POINTER or E_OUTOFMEMORY.

Remarks

This member function overrides the CBasePin::EnumMediaTypes member function and implements the IPin::EnumMediaTypes method. Transform-inplace filters use the media type enumerator from adjacent filters because they do not change the media type. When asked by a connected output pin of the upstream filter for this pin's media type enumerator, this member

function simply retrieves the allocator from the input pin connected to its output pin (if it is connected).

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::GetAllocator

CTransInPlaceInputPin Class

Retrieves the upstream allocator.

HRESULT GetAllocator(
IMemAllocator ** *ppAllocator*
);

Parameters

ppAllocator
Returned allocator.

Return Values

Returns a NOERROR if the method retrieves an allocator being used by the downstream filter. If no such allocator exists, returns S_OK if the method retrieves an allocator being used by the output pin of the in-place transform filter. If neither of these types of allocators can be retrieved, returns VFW_E_NO_ALLOCATOR.

Remarks

This member function overrides the CBaseInputPin::GetAllocator member function and implements the IMemInputPin::GetAllocator method. If an allocator has already been agreed upon, this member function supplies that allocator. Otherwise, if the downstream input pin can supply an allocator, it does so. If no allocator is available, this member function returns VFW_E_NO_ALLOCATOR.

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::GetAllocatorRequiremen

CTransInPlaceInputPin Class

Passes requests for allocator requirements downstream.

HRESULT GetAllocatorRequirements(
ALLOCATOR_PROPERTIES * *pProps*
);

Parameters

pProps
ALLOCATOR_PROPERTIES structure containing the required size, count, and alignment of the allocator.

Return Values

Returns E_NOTIMPL if the filter's output pin is not connected. Otherwise, returns an HRESULT that indicates whether the allocator properties were successfully received. **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 member function overrides the CBaseInputPin::GetAllocatorRequirements member function and implements the IMemInputPin::GetAllocatorRequirements method. If the downstream input pin can supply allocator requirements, it does so.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::NotifyAllocator

CTransInPlaceInputPin Class

Receives notification of which allocator will be used by the connected output pin.

HRESULT NotifyAllocator(
IMemAllocator * *pAllocator*,
BOOL *bReadOnly*

```
);
```

Parameters

pAllocator
Pointer to the IMemAllocator object to use. This might or might not be the same CTransInPlaceInputPin object that the input pin provided in the CTransInPlaceInputPin::GetAllocator member function (the output pin could provide its own allocator).

bReadOnly
Flag to indicate if the samples from this allocator are read-only.

Return Values

Returns NOERROR if successful. Returns E_POINTER if the pointer is invalid. Otherwise, returns an error due to calling CTransInPlaceOutputPin::ReceiveAllocator.

Remarks

This member function overrides the CBaseInputPin::NotifyAllocator member function and implements the IMemInputPin::NotifyAllocator method. This member function remembers the allocator and passes it to the output pin because they both must share the same allocator.

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::PeekAllocator

CTransInPlaceInputPin Class

Returns a pointer to the default allocator.

IMemAllocator * PeekAllocator( )

Return Values

Returns the m_pAllocator data member inherited from CBaseInputPin.

Remarks

This method does not increment the reference count.

Previous Home Topic Contents Index Next

CTransInPlaceInputPin::ReadOnly

CTransInPlaceInputPin Class

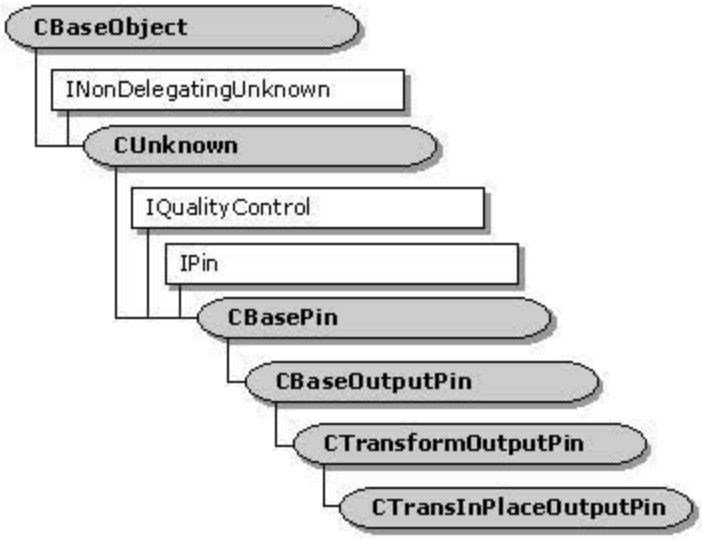
Returns m_bReadOnly to indicate whether or not a stream is read-only.

const BOOL ReadOnly( )

Return Values

Returns TRUE if the stream is read-only. Returns FALSE otherwise.

CTransInPlaceOutputPin Class



The **CTransInPlaceOutputPin** class implements the output pin of a simple transform-inplace filter (CTransInPlaceFilter).

Protected Data Members

Name	Description
m_pTIPFilter	Pointer to the CTransInPlaceFilter object that owns this pin.

Member Functions

Name	Description
ConnectedIMemInputPin	Returns a pointer to the input pin to which this output pin is connected.
CTransInPlaceOutputPin	Construct a CTransInPlaceOutputPin object.
PeekAllocator	Returns a pointer to the default allocator.
ReceiveAllocator	Receives notification of which allocator will be used.

Overridable Member Functions

Name	Description
CheckMediaType	Determines if the media type is acceptable.
DecideAllocator	Negotiates the allocator to use (uses the allocator from the upstream output pin).

Implemented IPin Methods

Name	Description
EnumMediaTypes	Provides a media type enumerator from the upstream filter.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::CheckMediaType

CTransInPlaceOutputPin Class

Determines if the media type is acceptable.

HRESULT CheckMediaType(
const CMediaType* *pmt*
);

Parameters

pmt
Pointer to a media type object containing the proposed media type.

Return Values

Returns S_OK if the pin is not connected. Otherwise, returns S_TRUE if the media type is accepted or S_FALSE if it is not.

Remarks

This member function overrides the CTransformOutputPin::CheckMediaType member function. It calls the pure virtual CTransformFilter::CheckInputType member function to verify the media type (which you must implement in your derived class) because it does not change the media type from input to output. If it is not connected, it returns S_OK, which agrees to any media type; otherwise, it calls QueryAccept on the output pin of the upstream filter and returns the result.

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::ConnectedIMemInputP

CTransInPlaceOutputPin Class

Returns a pointer to the input pin to which this output pin is connected.

IMemInputPin * ConnectedIMemInputPin( )

Return Values

Returns the m_pInputPin data member inherited from CBaseOutputPin.

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::CTransInPlaceOutputPi

CTransInPlaceOutputPin Class

Constructs a CTransInPlaceOutputPin object.

CTransInPlaceOutputPin(
TCHAR *pObjectName*,
CTransInPlaceFilter *pFilter*,
HRESULT * *phr*,
LPCWSTR *pName*
);

Parameters

pObjectName
Name of the CTransInPlaceOutputPin object.
pFilter
Pointer to the owning CTransInPlaceFilter filter.
phr
Pointer to an HRESULT value in which to return resulting information.
pName
Name of the pin.

Return Values

No return value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::DecideAllocator

CTransInPlaceOutputPin Class

Negotiates the allocator to use (uses the allocator from the upstream output pin).

HRESULT DecideAllocator(
IMemInputPin * *pPin*,
IMemAllocator ** *ppAlloc*
);

Parameters

pPin
Pointer to the IMemInputPin interface of the downstream input pin.
ppAlloc
Returned allocator pointer.

Return Values

Returns NOERROR if successful. Otherwise, returns VFW_E_NO_ALLOCATOR if there is no allocator, or an error from calling GetAllocator, InitAllocator, GetAllocatorRequirements, DecideBufferSize, or NotifyAllocator.

Remarks

This member function overrides the CBaseOutputPin::DecideAllocator member function. This implementation uses the allocator that is negotiated by its input pin because a transform-inplace filter does not supply its own allocator. It then calls IMemInputPin::NotifyAllocator on the downstream input pin with that allocator.

If you want to use your own allocator, it is better to derive from CTransformFilter than from CTransInPlaceFilter, because the purpose of a transform-inplace filter is to use an existing allocator.

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::EnumMediaTypes

CTransInPlaceOutputPin Class

Provides a media type enumerator from the upstream filter.

HRESULT EnumMediaTypes(
IEnumMediaTypes ***ppEnum**
);

Parameters

ppEnum
Pointer to an enumerator for the media types.

Return Values

Returns NOERROR if successful, VFW_E_NOT_CONNECTED if there is no connection, or an HRESULT that indicates an error with the enumerator, such as E_POINTER or E_OUTOFMEMORY.

Remarks

This member function overrides the CBasePin::EnumMediaTypes member function and implements the IPin::EnumMediaTypes method. Transform-inplace filters use the media type enumerator from adjacent filters because they do not change the media type. This member function calls **IPin::EnumMediaTypes** on the output pin connected to the filter's input pin. If an application receives an enumerator, the application must release it when finished.

Previous Home Topic Contents Index Next

CTransInPlaceOutputPin::PeekAllocator

CTransInPlaceOutputPin Class

Returns a pointer to the default allocator.

IMemAllocator * PeekAllocator( )

Return Values

Returns the m_pAllocator data member inherited from CBaseOutputPin.

Remarks

This member function does not increment the reference count.

CTransInPlaceOutputPin::ReceiveAllocator

CTransInPlaceOutputPin Class

Receives notification of which allocator will be used.

HRESULT ReceiveAllocator(
IMemAllocator * *pAllocator*,
BOOL *bReadOnly*
);

Parameters

pAllocator
Pointer to the IMemAllocator object to use.
bReadOnly
Flag to indicate if the samples from this allocator are read-only.

Return Values

Returns NOERROR if the allocator has the correct properties and is not read-only. Returns S_OK if successful if the allocator has the correct properties but is read-only; otherwise, returns VFW_E_BADALIGN, VFW_E_ALREADY_COMMITTED, VFW_E_BUFFERS_OUTSTANDING, or E_FAIL if the allocator's properties don't match what is needed.

Remarks

This member function is called by the CTransInPlaceInputPin::NotifyAllocator member function to indicate to the output pin which allocator will be used. It is only called if the output pin is connected. The choice is propagated to input pins downstream if the allocator is not read-only. For read-only allocators, only the properties are passed downstream.

CUnknown Class

All Microsoft® DirectShow™ Component Object Model (COM) objects derive from the **CUnknown** abstract base class. This class facilitates the creation of simple COM objects that you can combine with other COM objects to support multiple interfaces. To use this class, derive your object from **CUnknown** and call the DECLARE_IUNKNOWN macro in the public section of your object class definition; this implements the IUnknown interface for your object. Note that if derive from an object that has already done this, such as CBaseFilter, you do not need to do it yourself.

The **CUnknown** class supports only one interface, IUnknown. To support interfaces in addition to those provided by the base class, override the NonDelegatingQueryInterface method. In the overriding function, call the GetInterface function to retrieve the interface pointer for any interfaces your object supports. If the derived class does not implement the specified interface, you must query the base class to retrieve the interface.

For example, CBaseFilter supports the following interfaces directly.

- IBaseFilter
- IPersist
- IAMovieSetup

CBaseFilter also supports IUnknown by passing queries for this interface to **CUnknown**. The following code sample demonstrates this process.

```
/* Override this to say what interfaces are supported and where */

STDMETHODIMP CBaseFilter::NonDelegatingQueryInterface(REFIID riid, void **ppv)
{
    CheckPointer(ppv,E_POINTER);
    ValidateReadWritePtr(ppv,sizeof(PVOID));

    /* Do we have this interface */

    if (riid == IID_IFilter) {
         return GetInterface((IBaseFilter *) this, ppv);
    } else if (riid == IID_IMediaFilter) {
                  return GetInterface((IMediaFilter *) this, ppv);
    } else if (riid == IID_IPersist) {
                  return GetInterface((IPersist *) this, ppv);
    } else if (riid == IID_IAMovieSetup) {
                  return GetInterface((IAMovieSetup *) this, ppv);
```

```
    } else {
                return CUnknown::NonDelegatingQueryInterface(riid, ppv);
    }
}
```

To build composite objects, the **CUnknown** constructor has an LPUNKNOWN parameter that is a pointer to the top-level IUnknown interface for the entire composite object (the object that includes all objects based on a class derived from **CUnknown**). If this value is non-NULL, **CUnknown** stores a pointer to the topmost object; if it is null, the topmost object is **CUnknown** itself. This way, the topmost object's **IUnknown** has the same implementation as the INonDelegatingUnknown interface.

A derived class will typically override the NonDelegatingQueryInterface method to return interfaces that it supports; however, it must delegate support for IUnknown to the **CUnknown** class implementation. Usually NonDelegatingAddRef and NonDelegatingRelease do not need to be overridden because the reference count for the whole object is managed inside the top-level object. However, **NonDelegatingRelease** might need to be overridden sometimes because its default action when the reference count goes to zero is to delete the object from inside itself.

CUnknown provides the CUnknown::GetOwner member function. GetOwner simply returns an LPUNKNOWN pointer to the controlling unknown. This is used in the DECLARE_IUNKNOWN macro when calling QueryInterface. It can also be used when creating a composite object to pass an LPUNKNOWN pointer to a component interface as an (equivalent) alternative to passing the LPUNKNOWN pointer that was passed to the composite object constructor.

When QueryInterface is called on an interface owned by a component interface, it is immediately passed to the NonDelegatingQueryInterface method of the top-level object's INonDelegatingUnknown::NonDelegatingQueryInterface method, which either returns an interface it implements itself or passes the call to the correct member or base class's **INonDelegatingUnknown::NonDelegatingQueryInterface** method. This then repeats the process until a component is found that implements the interface or calls CUnknown::NonDelegatingQueryInterface, which fails the call.

Note that the top-level object's CUnknown::NonDelegatingQueryInterface member function (as distinct from its own implementation) must be called to support IUnknown.

This design makes support for COM aggregation straightforward. The derived object's **CreateInstance** member function, which is called from the class factory (by **CClassFactory::CreateInstance**) passes the outer unknown (the *pUnkOuter* parameter from CoCreateInstance) on to **CUnknown** by calling the class constructor. So the object behaves as if it were part of a larger object by delegating its QueryInterface calls to the outer unknown.

Protected Data Members

Name	Description
m_cRef	Number of reference counts (so the INonDelegatingUnknown::NonDelegatingRelease method can be overridden).

Member Functions

Name	Description
CUnknown	Constructs a CUnknown object.
GetOwner	Returns an LPUNKNOWN pointer to the controlling unknown.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingAddRef	Increments the reference count for an interface.
NonDelegatingQueryInterface	Returns an interface and increments the reference count.
NonDelegatingRelease	Decrements the reference count for an interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CUnknown::CUnknown

CUnknown Class

Constructs a CUnknown object.

CUnknown(
const TCHAR **pName*,
LPUNKNOWN *pUnk*
);

Parameters

pName
Name of the object used in the CBaseObject constructor for debugging purposes.

pUnk
Pointer to the owner of this object. If non-NULL, IUnknown calls are delegated to this object.

Return Values

No return value.

Remarks

The object is initialized with a reference count of zero. This reference count can be incremented when the object is queried for its first interface, depending on whether the object is currently being aggregated.

Previous Home Topic Contents Index Next

CUnknown::GetOwner

CUnknown Class

Retrieves this object's Component Object Model (COM) class owner.

LPUNKNOWN GetOwner(void);

Return Values

Returns an LPUNKNOWN pointer to the controlling IUnknown interface.

Previous Home Topic Contents Index Next

CUnknown::NonDelegatingAddRef

CUnknown Class

Increments the reference count for an interface.

ULONG NonDelegatingAddRef( );

Return Values

Returns the reference count of the object.

Remarks

This member function provides a base class implementation of the INonDelegatingUnknown::NonDelegatingAddRef method. When the object derived from CUnknown is part of an aggregated object, this reference count modification is private to the embedded object.

Previous Home Topic Contents Index Next

CUnknown::NonDelegatingQueryInterface

CUnknown Class

Returns an interface and increments the reference count.

HRESULT NonDelegatingQueryInterface(
REFIID *riid*,
void ** *ppv*
);

Parameters

riid
Reference identifier.
ppv
Pointer to the interface.

Return Values

Returns E_POINTER if *ppv* is invalid. Returns NOERROR if the query is successful or E_NOINTERFACE if it is not.

Remarks

This member function provides a base class implementation of the INonDelegatingUnknown::NonDelegatingQueryInterface method. Override this class to return interfaces on the object in the derived class.

Previous Home Topic Contents Index Next

CUnknown::NonDelegatingRelease

CUnknown Class

Decrements the reference count for an interface.

ULONG NonDelegatingRelease( );

Return Values

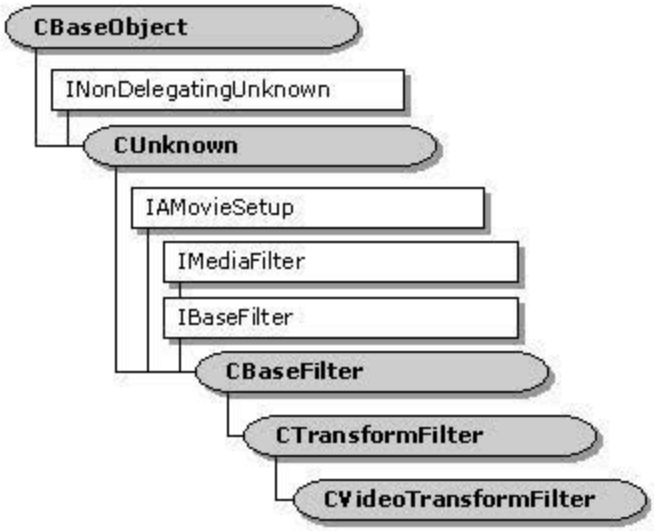
Returns the reference count.

Remarks

This member function provides a base class implementation of the INonDelegatingUnknown::NonDelegatingRelease method. When the object derived from CUnknown is part of an aggregated object, this reference count modification is private to the

embedded object.

CVideoTransformFilter Class



The **CVideoTransformFilter** class is designed primarily as a base class for AVI decompressor filters. It is based on a "copying" transform class and assumes that the output buffer will likely be a video buffer or Microsoft® DirectDraw® buffer, although this could be used as a base class for other types of transform filters. The main feature of this class is that it enables quality-control management in a transform filter. This means that it decides to drop frames based on receiving a quality notification from the renderer, and taking into account other factors about the media stream it is processing and the filter's own behavior.

Every time the CVideoTransformFilter::Receive member function is called, it calls CVideoTransformFilter::ShouldSkipFrame to determine whether to start, continue, or stop skipping frames. This member function starts skipping samples only if all the following conditions are true.

- The average time to decode is more than one fourth of the frame time.
- The filter is running at least one frame late.
- The next anticipated key frame is estimated to be no more than one frame early.
- The occurrence of key frames is sufficiently frequent.

Once the class starts to skip frames, it will skip all frames until a key frame appears, at which time it resets the m_bSkipping flag and processes the sample.

Key frames are defined as AVI key frames or MPEG I frames. These require no history to decode and, if they are skipped, no other frames can be decoded until the next key frame. Non-key frames include AVI non-key frames, MPEG P frames, and MPEG B frames. MPEG B frames are treated the same as other non-key frames by this class. (MPEG B frames can be dropped without the need to skip further frames; however, because this class is aimed primarily at AVI decompressors, it does not allow for this. Once any frame is skipped, all frames are skipped up to the next key frame.)

Protected Data Members

Name	Description
m_bNoSkip	Set to TRUE to prevent skipping to the next key frame (for debugging the filter).
m_bQualityChanged	Status flag that indicates if the stream has degraded. This is set to TRUE in CVideoTransformFilter::Receive if the call to the derived class Transform member function fails. (Receive returns NOERROR in this case because returning S_FALSE indicates that end-of-stream has arrived.)
m_bSkipping	Set to TRUE if the filter is skipping to the next key frame.
m_idFrameType	Performance-measuring frame type identifier (available if PERF is defined). Logs 1 for key frames; logs 2 for nonkey frames.
m_idLate	Performance identifier for measuring lateness (available if PERF is defined).
m_idSkip	Performance identifier for measuring frame skipping (available if PERF is defined).
m_idTimeTillKey	Performance identifier that represents an estimate of the time in milliseconds until the next key frame arrives (available if PERF is defined).
m_itrAvgDecode	Average time required to decode (transform) the sample. If this is less than one-fourth of the frame time, it is assumed the quality problems are not being generated by this filter and no frames are dropped.
m_itrLate	Amount of time that the current frame is late. This is originally set to the value of the Quality structure's Late member passed in the quality control message from the renderer filter. It is decremented by the frame time of each frame that is skipped.
m_nFramesSinceKeyFrame	Used to count frames since the last key frame.
m_nKeyFramePeriod	The largest observed interval between key frames.
m_nWaitForKey	Used to ensure output after a format change before getting the first key frame. When nonzero, frames are not passed to the renderer. Set to 30 when format is changed and decremented on each non-key frame.
m_tDecodeStart	Time since the start of the decoding.

Member Functions

Name	Description
AlterQuality	Receives a quality-control notification from the output pin and provides an opportunity to alter the quality of the media stream.
CVideoTransformFilter	Constructs a CVideoTransformFilter object.
ShouldSkipFrame	Determines if the filter should start, continue, or stop skipping frames.

Overridable Member Functions

Name	Description
EndFlush	Receives notification of leaving the flushing state and passes it downstream.
Receive	Receives the media sample and either skips the sample or transforms and delivers the media sample.
RegisterPerfId	Registers a performance measurement identifier.
StartStreaming	Overrides CTransformFilter::StartStreaming to reset the quality control information when streaming starts or flushing starts.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CVideoTransformFilter::AlterQuality

CVideoTransformFilter Class

Receives a quality-control notification and provides an opportunity to alter the quality of the media stream.

```
virtual HRESULT AlterQuality(
  Quality q
  );
```

Parameters

q

Quality-control notification message.

Return Values

This member function returns E_FAIL by default.

Remarks

This member function overrides the CTransformFilter::AlterQuality member function. It is called by the CTransformOutputPin::Notify member function before calling the CBaseInputPin::PassNotify member function to pass the quality control message upstream. This function sets the CVideoTransformFilter::m_itrLate data member to the value Quality structure's Late member so that the filter can determine whether to skip frames. It returns E_FAIL so that the renderer downstream will continue to handle quality control.

Previous Home Topic Contents Index Next

CVideoTransformFilter::CVideoTransformFilter

CVideoTransformFilter Class

Constructs a CVideoTransformFilter object.

CVideoTransformFilter(
TCHAR **pName*,
LPUNKNOWN *pUnk*,
REFCLSID *clsid*
);

Parameters

pName
Name given to the CVideoTransformFilter object.
pUnk
Pointer to LPUNKNOWN.
clsid
Class identifier of the CVideoTransformFilter class.

Return Values

No return value.

Previous Home Topic Contents Index Next

CVideoTransformFilter::EndFlush

CVideoTransformFilter Class

Receives notification that the filter is leaving the flushing state and passes it downstream.

HRESULT EndFlush( );

Return Values

Returns VFW_E_NOT_CONNECTED if the filter finds no input pin; otherwise, returns the value that the IPin::EndFlush method returns.

Remarks

This member function overrides the CTransformFilter::EndFlush member function to reset quality management information.

Previous Home Topic Contents Index Next

CVideoTransformFilter::Receive

CVideoTransformFilter Class

Receives the media sample and either skips the sample or transforms and delivers the media sample.

HRESULT Receive(
IMediaSample *pSample*
);

Parameters

pSample
Sample to deliver.

Return Values

Returns an HRESULT value.

Remarks

This member function overrides the CTransformFilter::Receive member function. Override only if you need more control of the process.

Previous Home Topic Contents Index Next

CVideoTransformFilter::RegisterPerfId

CVideoTransformFilter Class

Registers performance measurement identifiers.

virtual void RegisterPerfId( );

Return Values

No return value.

Remarks

By default, this member function registers the following performance identifiers.

Performance identifier	Registered string
m_idSkip	Video transform skip frame
m_idFrameType	Video transform frame type
m_idLate	Video transform lateness
m_idTimeTillKey	Video transform estd. time to next key

This member function also calls CTransformFilter::RegisterPerfId for its performance identifier.

Override this member function if you want to register performance measurement identifiers in the derived class. If you do this, be sure to register these as well. This member function is enabled only when PERF is defined.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CVideoTransformFilter::ShouldSkipFrame

CVideoTransformFilter Class

Determines if the filter should start, continue, or stop skipping frames.

BOOL ShouldSkipFrame(
IMediaSample * *pIn*
);

Parameters

pIn
Received sample to be transformed or skipped.

Return Values

Returns TRUE if the filter should skip this sample; otherwise, returns FALSE.

Remarks

This member function sets the m_bSkipping member variable to FALSE if the sample is a key frame (sync point) and returns FALSE. This stops any skipping that has started. This member function starts skipping samples (sets **m_bSkipping** to TRUE and returns TRUE) only if all of the following conditions are true.

- The average time to decode is more than one-fourth of the frame time.
- The filter is running at least one frame late.
- The next anticipated key frame is estimated to be no more than one frame early.
- The occurrence of key frames is sufficiently frequent.

This member function sends an EC_QUALITY_CHANGE notification when sample skipping starts. Once skipping starts, all samples are skipped until the next key frame arrives.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CVideoTransformFilter::StartStreaming

CVideoTransformFilter Class

Overrides CTransformFilter::StartStreaming to reset the quality control information when streaming starts or flushing starts.

virtual HRESULT StartStreaming( );

Return Values

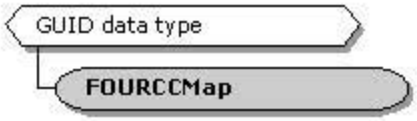
Returns NOERROR.

Remarks

This member function sets several quality control member variables to 0, including m_itrLate, m_nKeyFramePeriod, m_nFramesSinceKeyFrame, m_bSkipping, and m_tDecodeStart. It sets m_itrAvgDecode to 3000, and sets m_bQualityChanged and m_bSampleSkipped to FALSE.

Previous Home Topic Contents Index Next

FOURCCMap Class



This class provides conversion between GUID media subtypes and old-style FOURCC 32-bit media tags. In the original Microsoft® Windows® multimedia APIs, media types were tagged with 32-bit values created from four 8-bit characters and were known as **FOURCC**s. Microsoft DirectShow™ media types have **GUID**s for the subtype, partly because these are simpler to create (creation of a new **FOURCC** requires its registration with Microsoft). Because **FOURCC**s are unique, a one-to-one mapping has been made possible by allocating a range of 4,000 million **GUID**s representing **FOURCC**s. This range is all **GUID**s of the form:

```
XXXXXXXX-0000-0010-8000-00AA00389B71
```

This class simplifies conversion between GUIDs and FOURCCs. This is for compatibility only. It is recommended that all new media subtypes be represented by **GUID**s created by Guidgen.exe or a similar tool, and not by mapping **FOURCC**s.

The object is derived from a GUID, with no extra data members, and can be cast to a **GUID**. The object can be passed a FOURCC at construction time. The default constructor will initialize the **FOURCC** to zero.

The GetFOURCC and SetFOURCC methods do not check that the fixed portions of the GUID correspond to the FOURCC range. Thus, if you cast a pointer to a **GUID** into a pointer to a **FOURCC** and then set or get the **FOURCC** field, you also need to check separately that the **GUID** is within the **FOURCC** range.

Member Functions

Name	Description
FOURCCMap	Constructs a FOURCCMap object.
GetFOURCC	Returns the FOURCC from a FOURCCMap object.
SetFOURCC	Sets the FOURCC portion of the FOURCCMap object.

Previous Home Topic Contents Index Next

FOURCCMap::FOURCCMap

FOURCCMap Class

Constructs a FOURCCMap object. Provides a mapping between old-style multimedia format DWORD types and new-style GUID types.

FOURCCMap( );
FOURCCMap(
DWORD *Fourcc*
);
FOURCCMap(
const GUID * *pguid*
);

Parameters

Fourcc
DWORD media tag formerly used for Microsoft multimedia types.
pguid
Globally unique identifier (GUID).

Return Values

No return value.

Remarks

If this object is constructed with the FOURCC code, a GUID is created to match it. If this object is created with an existing **GUID**, the **FOURCC** value of the object is set to zero. Thereafter, the **FOURCC** value can be set or retrieved using the SetFOURCC and GetFOURCC member functions, respectively.

Previous Home Topic Contents Index Next

FOURCCMap::GetFOURCC

FOURCCMap Class

Retrieves the FOURCC DWORD from the FOURCCMap object.

DWORD GetFOURCC(void);

Return Values

Returns the FOURCC DWORD value. Note that if you construct a FOURCCMap object from a

GUID that was not originally derived from **FOURCC**, the return value will be essentially random.

Previous Home Topic Contents Index Next

FOURCCMap::SetFOURCC

FOURCCMap Class

Sets the FOURCC portion of the FOURCCMap object.

void SetFOURCC(
const GUID * *pguid*
);

Parameters

pguid
Pointer to the returned globally unique identifier (GUID) part of the FOURCCMap object.

Return Values

No return value.

Utility Functions

This section contains reference entries for the DirectShow utility functions and macros. DirectShow provides utilities for conversion, setup, timers, retrieving interfaces and declaring IUnknown, helper functions for math operations, property pages, BSTR functions, and strings, and stream integer functions. Most utilities are contained in Wxutil.h, but others are contained in Combase.h, Errors.h, Pstream.h, Refclock.h, Renbase.h, Videoctl.h, and Wtype.h.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

BSTR Functions

The Wxutil.h header file in the Microsoft® DirectShow™ base classes provides helper functions for allocating and freeing task-allocated BSTR strings.

Function	Description
FreeBSTR	Frees the task-allocated BSTR string.
WriteBSTR	Creates a task-allocated BSTR string by allocating task-allocated memory and copying a wide string to it.

Previous Home Topic Contents Index Next

FreeBSTR

BSTR Functions

Frees a task-allocated BSTR string from memory.

```
STDAPI FreeBSTR(
  BSTR* pstr
  );
```

Parameters

pstr
Address of the BSTR to free.

Return Values

Returns S_OK if successful, or S_FALSE if *pstr* is null.

Remarks

Memory is allocated for passing between objects across interfaces by calling CoTaskMemAlloc. It is freed by calling CoTaskMemFree. You can allocate, pass, and free memory safely between

objects created in different programming languages by using a central memory allocator.

WriteBSTR

BSTR Functions

Allocates and fills a task-allocated BSTR string.

STDAPI WriteBSTR(
BSTR* *pstrDest*,
LPCWSTR *szSrc*
);

Parameters

pstrDest
Pointer to where the address of the allocated BSTR will be stored.
szSrc
Wide (Unicode) string that will be copied to the newly allocated BSTR string.

Return Values

Returns an HRESULT value.

Remarks

Memory is allocated for passing between objects across interfaces by calling CoTaskMemAlloc. It is freed by calling CoTaskMemFree. By using a central memory allocator, memory can be allocated, passed, and freed safely between objects created in different programming languages.

Bitmap Functions, Macros, and Data

The Wxutil.h header file in the DirectShow base classes provides functions and macros to help convert between VIDEOINFOHEADER and BITMAPINFO structures.

Function	Description
BIT_MASKS_MATCH	Compares the masks of two video images.
BITMASKS	Retrieves a pointer to the array of bitmasks for the specified VIDEOINFOHEADER structure.
COLORS	Retrieves a pointer to an array of RGBQUAD structures that describes the color palette for the specified VIDEOINFOHEADER structure.
ContainsPalette	Checks if the video image contains a color palette.
DIBSIZE	Calculates the byte size of the specified bitmap
GetBitCount	Finds the number of bits per pixel.
GetBitmapFormatSize	Finds the size (in bytes) needed to build a VIDEOINFOHEADER structure and related data.
GetBitmapPalette	Finds the first palette entry in a VIDEOINFOHEADER structure.
GetBitmapSize	Finds the size (in bytes) needed to hold an image.
GetBitmapSubtype	Finds the GUID subtype for a given bitmap info header structure.
GetSubtypeName	Finds the (debug) name for a given GUID subtype.
GetTrueColorType	Finds the GUID subtype for a given bitmap header.
HEADER	Retrieves a pointer to the image data from the specified video image.
MPEG1_SEQUENCE_INFO	Retrieves the sequence header for the specified MPEG-1 video image.
PALETTISED	Checks if the video image's color palette is 8-bit or less.
PALETTE_ENTRIES	Returns the number of colors in the video image's palette.
RESET_MASKS	Clears the specified video image's bitmasks.
RESET_HEADER	Clears the specified video image.
RESET_PALETTE	Clears the specified video image's color palette.
SIZE_EGA_PALETTE	Calculates the size of the EGA (4-bit) color palette.
SIZE_MASKS	Calculates the size of the mask's color palette.
SIZE_MPEG1VIDEOINFO	Calculates the size of the specified MPEG-1 video image.
SIZE_PALETTE	Calculates the size of the 8-bit color palette.
SIZE_PREHEADER	Calculates the byte offset for the video image's bitmap information.
SIZE_VIDEOHEADER	Calculates the size of the video image.
TRUECOLORINFO	Retrieves a pointer to an array of TRUECOLORINFO structures that describes the bitmasks and color palette for the specified VIDEOINFOHEADER structure.

These functions are made available to help manage VIDEOINFOHEADER structures, which are used throughout DirectShow™ to describe video data streams. Although similar to the BITMAPINFO structure used in Microsoft® Win32® and existing multimedia, **VIDEOINFOHEADER** also adds some new video-specific fields.

Global Data	Description
bits555	Array of color bitmasks for an RGB 555 bitmap.
bits565	Array of color bitmasks for an RGB 565 bitmap.
bits888	Array of color bitmasks for an RGB 24-bit bitmap.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

BIT_MASKS_MATCH

Bitmap Functions, Macros, and Data

Retrieves the bitmasks for the specified video image.

BIT_MASKS_MATCH(
pbmi1,
pbmi2
)

Parameters

pbmi1
Pointer to a Win32 VIDEOINFOHEADER structure that contains the first video image.
pbmi2
Pointer to a Win32 VIDEOINFOHEADER structure that contains the second video image.

Return Values

Returns nonzero if the bitmasks for both video images are identical or zero otherwise.

Previous Home Topic Contents Index Next

BITMASKS

Bitmap Functions, Macros, and Data

Retrieves the bitmasks for the specified video image.

BITMASKS(
pbmi
)

Parameters

pbmi
Pointer to a Win32 VIDEOINFOHEADER structure that contains the video image.

Return Values

Returns a pointer to the array of bitmasks for the specified VIDEOINFOHEADER structure.

Previous Home Topic Contents Index Next

COLORS

Bitmap Functions, Macros, and Data

Retrieves the color palette for the specified video image.

COLORS(
pbmi
)

Parameters

pbmi
Pointer to a Win32 VIDEOINFOHEADER structure that contains the video image.

Return Values

Returns a pointer to an array of RGBQUAD structures that describes the color palette for the specified VIDEOINFOHEADER structure.

Previous Home Topic Contents Index Next

ContainsPalette

Bitmap Functions, Macros, and Data

Checks if the specified video image contains a color palette.

BOOL ContainsPalette(

const VIDEOINFOHEADER **pVideoInfo*
);

Parameters

pVideoInfo
Pointer to a VIDEOINFOHEADER structure.

Return Values

Returns TRUE if the VIDEOINFOHEADER structure contains a color palette or FALSE otherwise.

Previous Home Topic Contents Index Next

DIBSIZE

Bitmap Functions, Macros, and Data

Calculates the byte size of the specified bitmap.

DIBSIZE(
bi
)

Parameters

bi
A Win32 BITMAPINFOHEADER structure that specifies the source bitmap.

Return Values

Returns the byte size of the *bi* parameter.

Previous Home Topic Contents Index Next

GetBitCount

Bitmap Functions, Macros, and Data

Finds the number of bits per pixel.

WORD GetBitCount(
const GUID *pSubtype*
);

Parameters

pSubtype
Pointer to a GUID for a given video subtype.

Return Values

Returns the number of bits per pixel for this subtype, or USHRT_MAX if an error occurred.

Previous Home Topic Contents Index Next

GetBitmapFormatSize

Bitmap Functions, Macros, and Data

Finds the size (in bytes) needed to build a VIDEOINFOHEADER structure and related data.

LONG GetBitmapFormatSize(
const BITMAPINFOHEADER *pHeader*
);

Parameters

pHeader
Pointer to a Win32 BITMAPINFOHEADER structure.

Return Values

Returns the number of bytes for the VIDEOINFOHEADER structure described by this BITMAPINFOHEADER, including prefix information, the **BITMAPINFOHEADER** field, and any other color information on the end.

Previous Home Topic Contents Index Next

GetBitmapPalette

Bitmap Functions, Macros, and Data

Finds the first palette for a VIDEOINFOHEADER structure.

const RGBQUAD * GetBitmapPalette(
const VIDEOINFOHEADER *pVideoInfo*
);

Parameters

pVideoInfo
Pointer to a VIDEOINFOHEADER structure.

Return Values

Returns a pointer to the first entry in a palette.

Previous Home Topic Contents Index Next

GetBitmapSize

Bitmap Functions, Macros, and Data

Finds the number of bytes needed to hold an image.

DWORD GetBitmapSize(
const BITMAPINFOHEADER *pHeader*
);

Parameters

pHeader
Pointer to a Win32 BITMAPINFOHEADER structure.

Return Values

Returns the number of bytes needed to hold an image.

Previous Home Topic Contents Index Next

GetBitmapSubtype

Bitmap Functions, Macros, and Data

Finds the subtype for the specified bitmap.

```
const GUID GetBitmapSubtype(
  const BITMAPINFOHEADER *pHeader
  );
```

Parameters

pHeader
Pointer to a Win32 BITMAPINFOHEADER structure.

Return Values

Returns the video subtype GUID of the bitmap specified by *pHeader*, or GUID_NULL if *pHeader* is NULL.

Previous Home Topic Contents Index Next

GetSubtypeName

Bitmap Functions, Macros, and Data

Retrieves the name for a given GUID subtype.

```
TCHAR * GetSubtypeName(
  const GUID *pSubtype
  );
```

Parameters

pSubtype
Pointer to a GUID for a given video subtype.

Return Values

Returns the debug name of this GUID, or UNKNOWN if the name is not known.

GetTrueColorType

Bitmap Functions, Macros, and Data

Finds the subtype for the specified 16-bit color bitmap.

const GUID GetTrueColorType(
const BITMAPINFOHEADER **pHeader*
);

Parameters

pHeader
Pointer to a Win32 BITMAPINFOHEADER structure.

Return Values

Returns the video subtype GUID of the 16-bit color bitmap specified by *pHeader*, or GUID_NULL if *pHeader* is NULL.

HEADER

Bitmap Functions, Macros, and Data

Retrieves a pointer to the image data from the specified video image.

HEADER(
pVideoInfo
)

Parameters

pVideoInfo
Pointer to the VIDEOINFOHEADER structure that specifies the video image.

Return Values

Returns a pointer to the Win32 BITMAPINFOHEADER structure contained in the VIDEOINFOHEADER structure's bmiHeader data member.

Previous Home Topic Contents Index Next

MPEG1_SEQUENCE_INFO

Bitmap Functions, Macros, and Data

Retrieves the sequence header for the specified MPEG-1 video image.

MPEG1_SEQUENCE_INFO(
pv
)

Parameters

pv
Pointer to an MPEG1VIDEOINFO structure.

Return Values

Returns the bSequenceHeader data member of the specified MPEG1VIDEOINFO structure.

Previous Home Topic Contents Index Next

PALETTISED

Bitmap Functions, Macros, and Data

Checks if the video image's color palette is 8-bit or less.

PALETTISED(
pbmi
)

Parameters

pbmi
Pointer to the Win32 BITMAPINFOHEADER structure that specifies the video image.

Return Values

Returns nonzero if the video image's palette contains 256 or fewer colors, or zero otherwise.

Previous Home Topic Contents Index Next

PALETTE_ENTRIES

Bitmap Functions, Macros, and Data

Retrieves the number of colors in the video image's palette.

PALETTE_ENTRIES(
pbmi
)

Parameters

pbmi
Pointer to the Win32 BITMAPINFOHEADER structure that specifies the video image.

Return Values

Returns the number of colors in the video image's palette.

Previous Home Topic Contents Index Next

RESET_MASKS

Clears the specified video image's bitmasks.

RESET_MASKS(
pbmi

)

Parameters

pbmi
Pointer to the Win32 BITMAPINFOHEADER structure that specifies the video image.

Return Values

No return value.

Previous Home Topic Contents Index Next

RESET_HEADER

Bitmap Functions, Macros, and Data

Clears the specified video image.

RESET_HEADER(
pbmi
)

Parameters

pbmi
Pointer to the Win32 BITMAPINFOHEADER structure that specifies the video image.

Return Values

No return value.

Previous Home Topic Contents Index Next

RESET_PALETTE

Bitmap Functions, Macros, and Data

Clears the specified video image's color palette.

RESET_PALETTE(
pbmi
)

Parameters

pbmi
Pointer to the Win32 BITMAPINFOHEADER structure that specifies the video image.

Return Values

No return value.

Previous Home Topic Contents Index Next

SIZE_EGA_PALETTE

Bitmap Functions, Macros, and Data

Calculates the size of a 4-bit color palette.

SIZE_EGA_PALETTE

Return Values

Returns the size of a 16-color palette, in bytes.

Previous Home Topic Contents Index Next

SIZE_MASKS

Bitmap Functions, Macros, and Data

Calculates the size of a bitmask's color palette.

SIZE_MASKS

Return Values

Returns the size, in bytes, of a bitmap mask's color palette, which has three colors.

Previous Home Topic Contents Index Next

SIZE_MPEG1VIDEOINFO

Bitmap Functions, Macros, and Data

Calculates the size of the specified MPEG-1 video image.

SIZE_MPEG1VIDEOINFO(
pv
)

Parameters

pv
Pointer to the MPEG1VIDEOINFO structure that specifies the video image.

Return Values

Returns the byte size of the specified MPEG1VIDEOINFO structure.

Previous Home Topic Contents Index Next

SIZE_PALETTE

Bitmap Functions, Macros, and Data

Calculates the size of the 8-bit color palette.

SIZE_PALETTE

Return Values

Returns the size of the 256-color palette.

Previous Home Topic Contents Index Next

SIZE_PREHEADER

Bitmap Functions, Macros, and Data

Calculates the byte offset for the video image's bitmap information.

SIZE_PREHEADER

Return Values

Returns the byte offset of the VIDEOINFOHEADER structure's bmiHeader data member.

Previous Home Topic Contents Index Next

SIZE_VIDEOHEADER

Bitmap Functions, Macros, and Data

Calculates the size of the video image.

SIZE_VIDEOHEADER

Return Values

Returns the combined size of all of the VIDEOINFOHEADER structure's data members.

Previous Home Topic Contents Index Next

TRUECOLORINFO

Bitmap Functions, Macros, and Data

Retrieves the color palette and bitmasks for the specified video image.

TRUECOLORINFO(
pbmi
)

Parameters

pbmi
Pointer to a Win32 VIDEOINFOHEADER structure that contains the video image.

Return Values

Returns a pointer to an array of **TRUECOLORINFO** structures that describes the bitmasks and color palette for the specified VIDEOINFOHEADER structure.

Previous Home Topic Contents Index Next

CBaseRenderer Callback Function

The Renbase.h header file in the DirectShow base classes provides a function to signal the end of a stream in CBaseRenderer or its derived classes.

Name	Description
EndOfStreamTimer	Signals the end of the specified class's data stream.

Previous Home Topic Contents Index Next

EndOfStreamTimer

CBaseRenderer Callback Function

Signals the end of the specified class's data stream.

void CALLBACK EndOfStreamTimer(
UINT *uID*,
UINT *uMsg*,
DWORD *dwUser*,
DWORD *dw1*,
DWORD *dw2*

```
);
```

Parameters

uID
Integer value that specifies the timer value when the application called **EndOfStreamTimer**.
uMsg
Not used.
dwUser
DWORD value that contains the address of a class instance derived from CBaseRenderer.
dw1
Reserved.
dw2
Reserved.

Return Values

No return value.

Remarks

EndOfStreamTimer checks the m_EndOfStreamTimer data member of the class specified by *dwUser*. If **m_EndOfStreamTimer** is nonzero, **EndOfStreamTimer** sets it to zero and calls the class's SendEndOfStream method.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CCritSec Debug Functions

The Wxutil.h header file in the DirectShow base classes provides functions to make deadlocks easier to track. It is useful to insert an assertion in the code that says whether a critical section is owned or not. The routines that do the checking are global functions to avoid having different numbers of member functions in the debug and retail class implementations of CCritSec. In addition, Wxutil.h provides a routine that enables you to trace usage of specific critical sections. Because of the large number of critical sections, this assertion defaults to off.

Name	Description
CritCheckIn	Checks that the current thread is the owner of the given critical section.
CritCheckOut	Checks that the current thread is not the owner of the given critical section.
DbgLockTrace	Enables or disables debug logging of a given critical section.

Previous Home Topic Contents Index Next

CritCheckIn

CCritSec Debug Functions

Checks that the owner of *pcCrit* is the current thread.

BOOL WINAPI CritCheckIn(
CCritSec * *pcCrit*
);

Parameters

pcCrit
Pointer to a CCritSec critical section.

Return Values

Returns TRUE if the current thread is the owner of this critical section, or FALSE otherwise.

Remarks

If you call this function when DEBUG is not defined and you've included the DirectShow headers, it will always return TRUE.

Previous Home Topic Contents Index Next

CritCheckOut

CCritSec Debug Functions

Checks that the owner of *pcCrit* is not the current thread.

BOOL WINAPI CritCheckOut(
CCritSec * *pcCrit*
);

Parameters

pcCrit
Pointer to a CCritSec critical section.

Return Values

Returns TRUE if the current thread is not the owner of this critical section, or FALSE otherwise.

Remarks

If you call this function when DEBUG is not defined and you've included the DirectShow headers, it will always return TRUE.

Previous Home Topic Contents Index Next

DbgLockTrace

CCritSec Debug Functions

Enables or disables debug logging of a given critical section.

```
void WINAPI DbgLockTrace(
  CCritSec * pcCrit,
  BOOL fTrace
  );
```

Parameters

pcCrit
Pointer to a CCritSec critical section.
fTrace
Set to TRUE to enable logging or FALSE to disable it.

Return Values

No return value.

Remarks

This function does nothing unless DEBUG is defined when the DirectShow headers are included.

Previous Home Topic Contents Index Next

Conversion Functions

The Wxutil.h header file in the DirectShow base classes provides functions for converting between integers and wide strings.

Function	Description
atoi	Converts a string to an integer.
IntToWstr	Converts an integer to a wide string.
WstrToInt	Converts a wide string to an integer.

Previous Home Topic Contents Index Next

atoi

Conversion Functions

Converts a given string to an integer.

```
int WINAPI atoi(
  const TCHAR *sz
  );
```

Parameters

sz
Source character string.

Return Values

Returns the string's integer value.

Remarks

This version of **atoi** supports only decimal digits, and does not allow leading white space or signs. It supports both Unicode and ANSI strings. Other versions can vary.

Previous Home Topic Contents Index Next

IntToWstr

Conversion Functions

Converts a given integer value to a wide string representation.

void IntToWstr(
int *i*,
LPWSTR *wstrDest*
);

Parameters

i
Integer value to be converted.
wstrDest
LPWSTR that will contain the resulting wide string.

Return Values

No return value.

Previous Home Topic Contents Index Next

WstrToInt

Conversion Functions

Converts a given wide string value to an integer.

int WstrToInt(
LPCWSTR *wstrSrc*
);

Parameters

wstrSrc
Source wide-character string.

Return Values

Returns the string's integer value.

Previous Home Topic Contents Index Next

CPosPassThru Helper Function

The CPosPassThru helper function creates a plug-in distributor (a **CPosPassThru** COM object) that supports IMediaSeeking and IMediaPosition.

Function	Description
CreatePosPassThru	Creates a **CPosPassThru** COM object.

Previous Home Topic Contents Index Next

CreatePosPassThru

CPosPassThru Helper Function

Creates a COM object that supports IMediaSeeking and IMediaPosition for single-input pin renderers and transform filters.

```
STDAPI CreatePosPassThru(
  LPUNKNOWN pAgg,
  BOOL bRenderer,
  IPin *pPin,
  IUnknown **ppPassThru
  );
```

Parameters

pAgg
NULL if the object is not being created as part of an aggregate; otherwise, a pointer to the aggregate object's IUnknown interface (the controlling **IUnknown**).

bRenderer
TRUE if the filter supports rendering; otherwise, FALSE.

pPin
Pointer to the filter's input pin.

ppPassThru
ISeekingPassThru interface.

Return Values

Returns S_OK if successful; otherwise, returns an HRESULT indicating the error.

Remarks

You can use this function to create a CPosPassThru object in Quartz.dll rather than from your own .dll file. The CLSID of the object is CLSID_SeekingPassThru.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

DLL and Setup Functions

The Combase.h header file provides the following function for creating a run-time dynamic link with a specific dynamic-link library (DLL). For more information, read the *Run-Time Dynamic Linking* section in the Platform SDK.

Function	Description
LoadOLEAut32	Loads the Automation DLL (OleAut32.dll).

The Dllsetup.h header file provides the following functions for registering and unregistering DirectShow filters. You'll typically call AMovieDllRegisterServer2 to register your filter. The other functions are either helper functions or provide backwards compatibility.

Function	Description
AMovieDllRegisterServer	Registers filters. ActiveMovie 1.0 only.
AMovieDllRegisterServer2	Registers and unregisters filters.
AMovieDllUnregisterServer	Unregisters filters. ActiveMovie 1.0 only.
AMovieSetupRegisterFilter	Registers a filter's merit, pins, and media types in the registry using the filter mapper. ActiveMovie 1.0 only.
AMovieSetupRegisterFilter2	Registers a filter's merit, pins, and media types in the registry using **IFilterMapper2**.

Previous Home Topic Contents Index Next

LoadOLEAut32

DLL and Setup Functions

Loads the Automation dynamic-link library (OleAut32.dll).

HINSTANCE LoadOLEAut32( );

Return Values

Returns a handle to an Automation DLL instance.

Remarks

When the CBaseObject destructor destroys the object that loaded OleAut32.dll, it will unload the library if it is still loaded.

Previous Home Topic Contents Index Next

AMovieDllRegisterServer

DLL and Setup Functions

Registers filters. ActiveMovie 1.0 only.

HRESULT AMovieDllRegisterServer(void);

Return Values

Returns an HRESULT value.

Remarks

Use AMovieDllRegisterServer2 rather than this function to set up (register) your filters unless you need compatibility with ActiveMovie 1.0 filters. See Register DirectShow Objects and the sample filters included with the DirectShow SDK for more information about **AMovieDllRegisterServer2**.

Previous Home Topic Contents Index Next

AMovieDllRegisterServer2

DLL and Setup Functions

Registers and unregisters filters.

HRESULT AMovieDllRegisterServer2(
BOOL *bRegister*
);

Parameters

bRegister
TRUE indicates register the filter, FALSE indicates unregister it.

Return Values

Returns an HRESULT value.

Remarks

Use this function to set up your filters. See Register DirectShow Objects and the sample filters included with the DirectShow SDK for more information.

Note: The filter registration process is changing to allow filters to register by category. For example, capture filters and compression filters are enumerated together in their respective categories. The following functions demonstrate how filter registration and unregistration by category might work. The AMCap sample demonstrates this procedure. The following function uses the IFilterMapper2 interface.

```
// Register Sample Compressor Filter
STDAPI
DllRegisterServer( void )
{
 HRESULT hr = AMovieDllRegisterServer2( TRUE );
 if( FAILED(hr) )
     return hr;

 const WCHAR *wszUniq = L"Sample Compressor Filter" ;

 IFilterMapper2 *pFm2 = 0;

 hr = CoCreateInstance( CLSID_FilterMapper2
                        , NULL
                        , CLSCTX_INPROC_SERVER
                        , IID_IFilterMapper2
                        , (void **)&pFm2       );

 if(FAILED(hr))
     return hr;

 hr = pFm2->RegisterFilter(
      CLSID_SampleCompressorFilter,
      wszUniq,
      0,
      &CLSID_VideoCompressorCategory,
      wszUniq,
      MERIT_DO_NOT_USE,
      NULL,
```

```
      0);

 pFm2->Release();

 return hr;
}

// Unregister Sample Compressor Filter
STDAPI
DllUnregisterServer( void )
{

HRESULT hr = AMovieDllRegisterServer2( FALSE );
if( FAILED(hr) )
    return hr;

const WCHAR *wszUniq = L"Sample Compressor Filter" ;

IFilterMapper2 *pFm2 = 0;

hr = CoCreateInstance( CLSID_FilterMapper2
                       , NULL
                       , CLSCTX_INPROC_SERVER
                       , IID_IFilterMapper2
                       , (void **)&pFm2       );

if(FAILED(hr))
    return hr;

hr = pFm2->UnregisterFilter(
     &CLSID_VideoCompressorCategory,
     wszUniq,
     CLSID_SampleCompressorFilter);

pFm2->Release();

return hr;
}
```


Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AMovieDllUnregisterServer

DLL and Setup Functions

Unregisters filters. ActiveMovie 1.0 only.

HRESULT AMovieDllUnregisterServer(void);

Return Values

Returns an HRESULT value.

Remarks

Use AMovieDllRegisterServer2 rather than this function to uninstall (unregister) your filters unless you need compatibility with ActiveMovie 1.0 filters. See Register DirectShow Objects and the sample filters included with the DirectShow SDK for more information about **AMovieDllRegisterServer2**.

Previous Home Topic Contents Index Next

AMovieSetupRegisterFilter

DLL and Setup Functions

Registers a filter's merit, pins, and media types in the registry using the filter mapper. ActiveMovie 1.0 only.

```
HRESULT AMovieSetupRegisterFilter(
  const AMOVIESETUP_FILTER *const psetupdata,
  IFilterMapper *pIFM,
  BOOL bRegister
  );
```

Parameters

psetupdata
Pointer to the AMOVIESETUP_FILTER data.
pIFM
Pointer to IFilterMapper interface.
bRegister
TRUE indicates register the filter, FALSE indicates unregister it.

Return Values

Returns an HRESULT value.

Remarks

The CBaseFilter base class uses this helper function to register a filter if the 1.0 ActiveMovie runtime is installed. It is provided for compatibility with ActiveMovie version 1.0 only.

Typically a filter will use AMovieDllRegisterServer and will not call this function directly.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AMovieSetupRegisterFilter2

DLL and Setup Functions

Registers a filter's merit, pins, and media types in the registry using **IFilterMapper2**.

```
HRESULT AMovieSetupRegisterFilter2(
  const AMOVIESETUP_FILTER *const psetupdata,
  IFilterMapper2 *pIFM2,
  BOOL bRegister
  );
```

Parameters

psetupdata
Pointer to the AMOVIESETUP_FILTER data.
pIFM
Pointer to **IFilterMapper2** interface.
bRegister
TRUE indicates register the filter, FALSE indicates unregister it.

Return Values

Returns an HRESULT value.

Remarks

AMovieDllRegisterServer2 uses this helper function to register a filter after the COM server has been registered.

Typically a filter will use AMovieDllRegisterServer2 and will not call this function directly.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Error Message Function

The Errors.h header file provides a function for getting an error message for a given message identifier in the current language. The same header also provides the MAX_ERROR_TEXT_LEN equate, which indicates the maximum number of characters allowed in a message.

Function	Description
AMGetErrorText	Gets the error message text for a given message ID.

Previous Home Topic Contents Index Next

AMGetErrorText

Error Message Function

Retrieves the error message text for a given message identifier in the appropriate language.

```
DWORD AMGetErrorText(
  HRESULT hr,
  TCHAR *pBuffer,
  DWORD MaxLen
  );
```

Parameters

hr
Message identifier for the message text to be returned.

pBuffer
Area into which the message text will be stored.

MaxLen
Number of characters that *pBuffer* points to.

Return Values

Returns the number of characters stored in the buffer, or zero if an error occurred.

Previous Home Topic Contents Index Next

IUnknown Macro

To simplify the creation of new interfaces, the Combase.h header file includes a macro that

declares the three methods of the IUnknown interface.

Function	Description
DECLARE_IUNKNOWN	Declares the three methods of the base interface for a new interface.

Previous Home Topic Contents Index Next

DECLARE_IUNKNOWN

IUnknown Macro

Declares the three methods of the base interface for a new interface.

#define DECLARE_IUNKNOWN

Remarks

When you create a new interface, it must derive from IUnknown, which has three methods: QueryInterface, AddRef, and Release. This macro simplifies the declaration process by declaring each of these methods for the new interface.

Previous Home Topic Contents Index Next

INonDelegatingUnknown Interface

To enable a class to support both nondelegating and delegating IUnknown interfaces in the same COM object, the Combase.h header file declares the INonDelegatingUnknown interface. This interface is a version of **IUnknown** and has three methods:

INonDelegatingUnknown::NonDelegatingQueryInterface

INonDelegatingUnknown::NonDelegatingAddRef

INonDelegatingUnknown::NonDelegatingRelease

For sample implementations of these methods, see CUnknown::NonDelegatingQueryInterface, CUnknown::NonDelegatingAddRef, and CUnknown::NonDelegatingRelease.

Previous Home Topic Contents Index Next

INonDelegatingUnknown

INonDelegatingUnknown Interface

A version of IUnknown renamed to enable a class to support both nondelegating and delegating **IUnknown** interfaces in the same COM object. The interface supports the following three methods, in vtable order:

HRESULT NonDelegatingQueryInterface(
REFIID *iid*,
void** *ppvObject* **);**

ULONG NonDelegatingAddRef(void);

ULONG NonDelegatingRelease(void);

Remarks

To use **INonDelegatingUnknown** for multiple inheritance, perform the following steps:

1. Derive your class from an interface, for example, IMyInterface.
2. Include DECLARE_IUNKNOWN in your class definition to declare implementations of **QueryInterface**, **AddRef**, and **Release** that call the outer unknown.
3. Override **NonDelegatingQueryInterface** to expose IMyInterface with code such as the following:

```
if (riid == IID_IMyInterface) {
    return GetInterface((IMyInterface *) this, ppv);
} else {
    return CUnknown::NonDelegatingQueryInterface(riid, ppv);
}
```

4. Declare and implement the member functions of IMyInterface.

To use **INonDelegatingUnknown** for nested interfaces, perform the following steps:

1. Declare a class derived from CUnknown.
2. Include DECLARE_IUNKNOWN in your class definition.
3. Override **NonDelegatingQueryInterface** to expose IMyInterface with the code such as the following:

```
if (riid == IID_IMyInterface) {
    return GetInterface((IMyInterface *) this, ppv);
} else {
    return CUnknown::NonDelegatingQueryInterface(riid, ppv);
}
```

4. Implement the member functions of IMyInterface. Use CUnknown::GetOwner to access the COM object class.
5. In your COM object class, make the nested class a friend of the COM object class, and declare an instance of the nested class as a member of the COM object class.

   Because you must always pass the outer unknown and an **HRESULT** to the CUnknown constructor, you can't use a default constructor. You have to make the member variable a pointer to the class and make a new call in your constructor to actually create it.

6. Override the **NonDelegatingQueryInterface** with code such as the following:

```
if (riid == IID_IMyInterface) {
    return m_pImplFilter->
       NonDelegatingQueryInterface(IID_IMyInterface, ppv);
} else {
    return CUnknown::NonDelegatingQueryInterface(riid, ppv);
}
```

You can have mixed classes that support some interfaces through multiple inheritance and some interfaces through nested classes.

See Also

GetInterface, CUnknown, IUnknown Macro

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Math Helper Functions

The Wxutil.h header file in the DirectShow base classes provides some mathematical helper functions. These are intended to help with time format conversions.

Function	Description
llMulDiv	Implements ((a*b)+rnd)/c for 32-bit values of *a*.
Int64x32Div32	Implements ((a*b)+rnd)/c for 64-bit values of *a*.

Previous Home Topic Contents Index Next

llMulDiv

Math Helper Functions

Multiplies *a* by *b*, adds *rnd* to the 128-bit result, then divides by *c*.

```
LONGLONG WINAPI llMulDiv(
  LONGLONG a,
  LONGLONG b,
  LONGLONG c,
  LONGLONG rnd
  );
```

Return Values

Returns either the (a * b + rnd)/c calculation or one of the following values.

Value	Condition
0x7FFFFFFFFFFFFFFF	Overflow occurred because the result is too large (positive).
0x8000000000000000	Overflow occurred because the result is too large (negative).

Remarks

Rounding on the division is toward zero. Division by zero is counted as an overflow condition.

Previous Home Topic Contents Index Next

Int64x32Div32

Math Helper Functions

Multiplies *a* by *b*, adds *rnd* to the 96-bit result, then divides by *c*.

```
LONGLONG WINAPI Int64x32Div32(
  LONGLONG a,
  LONG b,
  LONG c,
  LONG rnd
  );
```

Return Values

Returns either the (a * b + rnd)/c calculation or one of the following values.

Value	Condition
0x7FFFFFFFFFFFFFFF	Overflow occurred because the result is too large (positive).
0x8000000000000000	Overflow occurred because the result is too large (negative).

Remarks

Rounding on the division is toward zero. Division by zero is counted as an overflow condition.

Previous Home Topic Contents Index Next

Media Type Functions

The Mtype.h header file in the DirectShow base classes provides helper functions for handling media types. These general-purpose functions create, copy and delete a task-allocated AM_MEDIA_TYPE structure. This is useful when using the IEnumMediaTypes interface, because the implementation allocates the structures that must be deleted later.

The functions are paired as follows:

- CreateMediaType is the opposite of DeleteMediaType.
- FreeMediaType is the opposite of CopyMediaType.

Function	Description
AreEqualVideoTypes	Compares the format, height, and width of two video sources.
CopyMediaType	Copies a task-allocated AM_MEDIA_TYPE structure.
CreateAudioMediaType	Initializes a media type structure given a wave format structure.
CreateMediaType	Allocates and initializes an AM_MEDIA_TYPE structure.
DeleteMediaType	Deletes a task-allocated AM_MEDIA_TYPE structure.
FreeMediaType	Frees a task-allocated AM_MEDIA_TYPE structure from memory.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

AreEqualVideoTypes

Media Type Functions

Determines if two media types have the same video format, width, and height.

BOOL WINAPI AreEqualVideoTypes (
CMediaType **pmt1*,
CMediaType **pmt2*
);

Parameters

pmt1
First media type to compare.
pmt2
Second media type to compare.

Return Values

Returns TRUE if *pmt1* and *pmt2* have the same video format, width, and height or FALSE otherwise.

Previous Home Topic Contents Index Next

CopyMediaType

Media Type Functions

Copies a task-allocated AM_MEDIA_TYPE structure.

void WINAPI CopyMediaType(
AM_MEDIA_TYPE **pmtTarget*,
const AM_MEDIA_TYPE **pmtSource*
);

Parameters

pmtTarget
Pointer to an area of memory in which to place the new copy of the structure.
pmtSource
Pointer to a source structure to copy.

Return Values

No return value.

Remarks

Free the resources in the *pmtTarget* structure by calling FreeMediaType when your code is done with the structure.

Previous Home Topic Contents Index Next

CreateAudioMediaType

Media Type Functions

Initializes a media type structure given a wave format structure.

```
STDAPI CreateAudioMediaType(
  const WAVEFORMATEX *pwfx,
  AM_MEDIA_TYPE *pmt,
  BOOL bSetFormat
  );
```

Parameters

pwfx
Pointer to the supplied WAVEFORMATEX structure.

pmt
Pointer to the AM_MEDIA_TYPE structure to initialize.

bSetFormat
Flag indicating whether to initialize the format section of the AM_MEDIA_TYPE structure, specifically the cbFormat and pbFormat members. Specify TRUE to initialize the format section, FALSE otherwise.

Return Values

Returns E_OUTOFMEMORY if memory could not be allocated for the format data; S_OK otherwise.

Previous Home Topic Contents Index Next

CreateMediaType

Media Type Functions

Creates a task-allocated AM_MEDIA_TYPE structure.

AM_MEDIA_TYPE * WINAPI CreateMediaType(
AM_MEDIA_TYPE const *pSrc*
);

Parameters

pSrc
Pointer to an AM_MEDIA_TYPE source structure.

Return Values

Returns a new AM_MEDIA_TYPE structure, or NULL if there is an error.

Remarks

Free the structure and resources allocated by this routine by calling DeleteMediaType when your code is done with the structure.

Previous Home Topic Contents Index Next

DeleteMediaType

Media Type Functions

Deletes a task-allocated AM_MEDIA_TYPE structure.

void WINAPI DeleteMediaType(
AM_MEDIA_TYPE *pmt*
);

Parameters

pmt
Pointer to an AM_MEDIA_TYPE structure.

Return Values

No return value.

Remarks

The structure should have been created by a call to CreateMediaType.

FreeMediaType

Media Type Functions

Frees a task-allocated AM_MEDIA_TYPE structure from memory.

void WINAPI FreeMediaType(
AM_MEDIA_TYPE& *mt*
);

Parameters

mt
Address of the structure.

Return Values

No return value.

Remarks

The structure should have been initialized by a call to CopyMediaType.

Object and Pin Functions

The Wxutil.h and Combase.h header files in the DirectShow base classes provides helper functions for comparing objects and pins and retrieving interfaces to objects.

Function	Description
EqualPins	Checks if two pins are on the same object.
GetInterface	Returns an interface pointer to the requested client.
IsEqualObject	Checks if two interfaces are on the same object.

Previous Home Topic Contents Index Next

EqualPins

Object and Pin Functions

Checks if two pins are on the same object.

```
BOOL EqualPins(
  IUnknown * pPin1,
  IUnknown * pPin2
  );
```

Parameters

pPin1
Address of one pin.
pPin2
Address of the other pin.

Return Values

Returns TRUE if the pins are both on the same object, or FALSE otherwise.

Previous Home Topic Contents Index Next

GetInterface

Object and Pin Functions

Retrieves an interface pointer.

```
HRESULT GetInterface(
  LPUNKNOWN pUnk,
  void **ppv
  );
```

Parameters

pUnk

Pointer to the IUnknown interface.

ppv

Retrieved interface.

Return Values

Returns an HRESULT value.

Remarks

This member function performs a thread-safe increment of the reference count. To retrieve the interface and add a reference, call this function from your overriding implementation of the INonDelegatingUnknown::NonDelegatingQueryInterface method.

Previous Home Topic Contents Index Next

IsEqualObject

Object and Pin Functions

Checks if two interfaces are on the same object.

```
BOOL WINAPI IsEqualObject(
  IUnknown * pFirst,
  IUnknown * pSecond
  );
```

Parameters

pFirst

Address of one interface.

pSecond

Address of the other interface.

Return Values

Returns TRUE if the interfaces are both on the same object, or FALSE otherwise.

Previous Home Topic Contents Index Next

Performance Macros

The Measure.h header file in the DirectShow base classes provides macros that help record performance data by maintaining a circular log of the start and stop times of certain events.

Macro	Description
MSR_START	Records the start time of the event.
MSR_STOP	Records the stop time of the event.

Previous Home Topic Contents Index Next

MSR_START

Performance Macros

Records the start time of the event with the given registered ID by adding the start time to the circular log and recording the time in *StatBuffer*.

```
#define MSR_START(
  int Id
  )
```

Parameters

Id

Registered ID of the event whose start is to be recorded.

Remarks

This macro does not update the statistical information. That happens when MSR_STOP is called.

Previous Home Topic Contents Index Next

MSR_STOP

Performance Macros

Records the stop time of the event with the given registered ID by adding the stop time to the circular log, and adding a StopTime-StartTime entry to the statistical record *StatBuffer*.

```
#define MSR_STOP(
  int Id
  )
```

Parameters

Id

Registered ID of the event whose stop is to be recorded.

Previous Home Topic Contents Index Next

Property Page Helper Functions

The Videoctl.h header file in the DirectShow base classes provides functions to help with property page implementations.

Function	Description
GetDialogSize	Retrieves the size of a resource dialog box in screen pixels.
StringFromResource	Loads a string from a resource file with the given resource identifier.
WideStringFromResource	Loads a Unicode string from a resource file with the given resource identifier.

Previous Home Topic Contents Index Next

GetDialogSize

Property Page Helper Functions

Retrieves the size of a resource dialog box.

```
BOOL WINAPI GetDialogSize(
  int iResourceID,
  DLGPROC pDlgProc,
  LPARAM lParam,
  SIZE *pResult
  );
```

Parameters

iResourceID
Dialog box resource identifier.
pDlgProc
Pointer to the dialog box procedure.
lParam
Any user data wanted in *pDlgProc*.
pResult
Size of the dialog box, in screen pixels.

Return Values

Returns TRUE if the dialog box resource was found, or FALSE otherwise.

Remarks

Property pages can use this function to return the actual display size they require. Most property pages are dialog boxes and, as such, have dialog box templates stored in resource files. Templates use dialog box units that do not map directly onto screen pixels. When a property page has its GetPageInfo function called, it must return the actual display size in pixels. This method is passed the resource ID for the dialog box and will return its size in pixels.

To make the calculation, the function creates an instance of the dialog box. To avoid the dialog box appearing on the screen temporarily, the dialog box's template in the resource file should not have a WS_VISIBLE property.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

StringFromResource

Property Page Helper Functions

Loads a string from a resource file with the given resource identifier.

```
TCHAR * WINAPI StringFromResource(
  TCHAR *pBuffer,
  int iResourceID
  );
```

Parameters

pBuffer
String corresponding to *iResourceID*.

iResourceID
Resource identifier of the string to retrieve.

Return Values

Returns the same string as *pBuffer*. If the function is not successful, returns a null string.

Remarks

The *pBuffer* buffer must be at least STR_MAX_LENGTH bytes.

Previous Home Topic Contents Index Next

WideStringFromResource

Property Page Helper Functions

Loads a Unicode string from a resource file with the given resource identifier.

```
WCHAR * WINAPI WideStringFromResource(
  WCHAR *pBuffer,
  int iResourceID
  );
```

Parameters

pBuffer
String corresponding to *iResourceID*.
iResourceID
Resource identifier of the string to retrieve.

Return Values

Returns the same string as *pBuffer*. If the function is not successful, returns a null string.

Remarks

Property pages are typically called through their COM interfaces, which use Unicode strings regardless of how the binary is built. This function allows you to convert a resource string to a Unicode string. The function converts the resource to a Unicode string (if it is not already one) after loading it.

Previous Home Topic Contents Index Next

Reference Time Function

The Refclock.h header file in the DirectShow base classes provides a reference time conversion function.

Function	Description
ConvertToMilliseconds	Converts the reference time to milliseconds.

Previous Home Topic Contents Index Next

ConvertToMilliseconds

Reference Time Function

Converts the reference time to milliseconds.

```
LONGLONG WINAPI ConvertToMilliseconds(
  const REFERENCE_TIME& RT
  );
```

Parameters

RT
Reference time, in 100-nanosecond units.

Return Values

Returns the reference time converted to milliseconds.

Previous Home Topic Contents Index Next

Stream Integer Functions

The Pstream.h header file in the DirectShow base classes contains a set of stream integer functions. These functions encode an integer in a stream object as 11 Unicode characters followed by one Unicode space. The interface to these functions might truncate to 32 bits.

Values such as (unsigned) 0x80000000 would be written as -2147483648, but would still load as 0x80000000 again through ReadInt.

Member Function	Description
WriteInt	Writes an integer to a stream encoded as a Unicode string.
ReadInt	Reads a Unicode string-encoded integer from a stream.

Previous Home Topic Contents Index Next

WriteInt

Stream Integer Functions

Writes an integer to an IStream, encoded as described in Stream Integer Functions.

```
STDAPI WriteInt(
  IStream *pIStream,
  int n
  );
```

Parameters

pIStream
Pointer to an IStream to which the encoded integer is to be written.

n
Integer value to be written.

Return Values

Returns an HRESULT value.

Remarks

The ReadInt function can be used to read the value written by **WriteInt**.

Previous Home Topic Contents Index Next

ReadInt

Stream Integer Functions

Reads an integer from an IStream.

STDAPI_(
int
)ReadInt(
IStream **pIStream*,
HRESULT *&hr*
);

Parameters

pIStream
Pointer to an IStream from which the encoded integer is to be read.
hr
Reference to an HRESULT value (output).

Return Values

Returns the integer value (truncated to 32 bits), or zero if an error occurred.

Remarks

This function is a stripped-down subset of what sscanf can do (without dragging in the C run time).

The **ReadInt** function can be used to read the value written by WriteInt.

Previous Home Topic Contents Index Next

String Functions

The Wxutil.h header file in the DirectShow base classes provides wide string functions, if they are not already provided by the Microsoft® Win32® environment.

Function	Description
AMGetWideString	Allocates and creates a wide string version of an existing nonwide string.
lstrcmpiW	Compares two wide strings, ignoring case.
lstrcmpW	Compares two wide strings.
lstrcpynW	Copies one wide string to another, with a maximum length.
lstrcpyW	Copies one wide string to another.
lstrlenW	Gets the length of a wide string in wide characters.

Previous Home Topic Contents Index Next

AMGetWideString

String Functions

Allocates and creates a Unicode version of an existing non-Unicode string.

STDAPI AMGetWideString(
LPCWSTR *pszString*,
LPWSTR **ppszReturn*
);

Parameters

pszString
Non-Unicode source string.
ppszReturn
Address of a Unicode string that will contain *pszString*.

Return Values

Returns S_OK if successful, E_POINTER if *ppszReturn* is NULL, or E_OUTOFMEMORY if not enough memory is available.

Previous Home Topic Contents Index Next

lstrcmpiW

String Functions

Compares two wide-character strings. The comparison is not case sensitive.

int lstrcmpiW(
LPCWSTR *lpszString1*,
LPCWSTR *lpszString2*
);

Parameters

lpszString1
Pointer to the first null-terminated wide string to be compared.
lpszString2

Pointer to the second null-terminated wide string to be compared.

Return Values

Returns a negative value if the function succeeds and the string that *lpszString1* points to is less than the string that *lpszString2* points to. Returns a positive value if the string that *lpszString1* points to is greater than the string that *lpszString2* points to. Returns zero if the strings are equal.

Remarks

The **lstrcmpiW** function compares two wide strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

The function returns the difference of the values of the first unequal characters it encounters. For instance, **lstrcmpiW** determines that L"abcz" is greater than L"abcdefg" and returns the difference of L'z' and L'd'.

The language (locale) is treated as always being English.

Previous Home Topic Contents Index Next

lstrcmpW

String Functions

Compares two wide-character strings. The comparison is case sensitive.

```
int lstrcmpW(
  LPCWSTR lpszString1,
  LPCWSTR lpszString2
  );
```

Parameters

lpszString1
Pointer to the first null-terminated wide string to be compared.
lpszString2
Pointer to the second null-terminated wide string to be compared.

Return Values

Returns a negative value if the function succeeds and the string that *lpszString1* points to is less than the string that *lpszString2* points to. Returns a positive value if the string that *lpszString1* points to is greater than the string that *lpszString2* points to. Returns zero if the

strings are equal.

Remarks

The **lstrcmpW** function compares two wide strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

The function returns the difference of the values of the first unequal characters it encounters. For instance, **lstrcmpW** determines that L"abcz" is greater than L"abcdefg" and returns the difference of L'z' and L'd'.

The language (locale) is treated as always being English.

Previous Home Topic Contents Index Next

lstrcpyW

String Functions

Copies a wide string to a buffer.

```
LPWSTR lstrcpyW(
  LPWSTR lpszString1,
  LPCWSTR lpszString2
  );
```

Parameters

lpszString1
Pointer to a buffer to receive the contents of the string pointed to by the *lpszString2* parameter. The buffer must be large enough to contain the string, including the terminating wide null character.

lpszString2
Pointer to the null-terminated wide string to be copied.

Return Values

Returns a pointer to the buffer.

Previous Home Topic Contents Index Next

lstrcpynW

String Functions

Copies a wide string to a buffer, up to a specified number of wide characters.

```
LPWSTR lstrcpynW(
  LPWSTR lpszString1,
  LPCWSTR lpszString2,
  int iMaxLength
  );
```

Parameters

lpszString1
Pointer to a buffer to receive the contents of the string that the *lpszString2* parameter points to. The buffer must be large enough to contain the string, including the terminating wide null character.

lpszString2
Pointer to the null-terminated wide string to be copied.

iMaxLength
Maximum number of wide characters to copy, including a terminating null character.

Return Values

Returns a pointer to the buffer.

Remarks

If *iMaxLength* is nonzero, **lstrcpynW** always inserts a terminating null wide character in the destination string, which could result in the source string being truncated.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

lstrlenW

String Functions

Retrieves the length of the specified wide string.

```
int lstrlenW(
  LPCWSTR lpszString
  );
```

Parameters

lpszString
Pointer to a null-terminated wide string.

Return Values

If the function succeeds, the return value specifies the length of the string, in wide characters.

Previous Home Topic Contents Index Next

Message Function

The Wxutil.h header file in the DirectShow base classes provides a helper function for processing messages.

Function	Description
WaitDispatchingMessages	Waits for a for the HANDLE *hObject* before dispatching messages. While waiting, messages sent to windows on the thread by SendMessage will be processed.

Previous Home Topic Contents Index Next

WaitDispatchingMessages

Message Function

Waits for a for the HANDLE *hObject* before dispatching messages. While waiting, messages sent to windows on the thread by SendMessage will be processed.

```
DWORD WINAPI WaitDispatchingMessages(
  HANDLE hObject,
  DWORD dwWait,
  HWND hwnd = NULL,
  UINT uMsg = 0
  );
```

Parameters

hObject
Handle of object to wait for.
dwWait
Time-out interval in milliseconds.
hwnd
Handle to a window.
uMsg
Win32 message.

Return Values

If the function succeeds, the return value indicates the event that caused the function to return. If the function fails, the return value is WAIT_FAILED.

The return value on success is one of the following values:

Value	Meaning
WAIT_ABANDONED	The specified object is a mutex (mutual exclusion) object that was not released by the thread that owned the mutex object before the owning thread terminated. Ownership of the mutex object is granted to the calling thread, and the mutex is set to nonsignaled.
WAIT_OBJECT_0	The state of the specified object is signaled.
WAIT_TIMEOUT	The time-out interval elapsed, and the object's state is nonsignaled.

Remarks

This function enables sent messages to be processed while waiting for a handle to a window. Use this function to wait for an object to be processed and to perform mutually exclusive operations, consequently avoiding possible deadlocks in objects with windows.

This helper function is similar to the Win32 WaitForSingleObject function.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Debugging

This section describes how to debug DirectShow in C and C++. It also contains reference entries for the macros and functions that DirectShow supplies to assist debugging.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Debugging with DirectShow

This article discusses debugging practices in Microsoft® DirectShow™ for the C and C++ languages. Many of these practices apply both to writing filters and to writing applications that use the DirectShow run time. The article provides some tips on writing code that can be easily debugged and some general debugging topics. This article also provides some hints about detecting memory leaks.

Contents of this article:

Writing Code You Can Test and Debug

Debugging code in the DirectShow environment can be easier if it's written to be easily tested and debugged in the first place. Some techniques that DirectShow supports include the following, which are discussed in this section.

Assertion Checking

Use assertion checking liberally. If you're not familiar with asserts, they're a popular way to isolate potential programming errors. DirectShow provides a number of assertion macros and functions, including ASSERT. The Microsoft® Foundation Classes (MFC) have an equivalent **ASSERT** macro. For example the following displays a message box if the value of *First* does not equal NULL:

```
ASSERT( First != NULL );
```

For more information about assertion, see Assert Macros and Functions.

Pass Debugging Names

Pass the debugging name to the constructors that support it. Tracking object creation and destruction is provided in debugging builds for the CBaseObject class and classes derived from it. The *object register* is the list of objects that have been created but not yet destroyed in those classes. The debugging name that is passed to the constructors of those classes is stored in the object register. For more information about debugging object registers and the DbgDumpObjectRegister function, see Object Register Debugging.

Debug Logging

Use the DirectShow DbgLog function to display debugging messages on a debugger as your program executes. Here's an example from the bouncing ball source filter:

```
DbgLog(( LOG_TRACE, 1, TEXT("New time: %d, Proportion: %d"),
    m_iRepeatTime, q.Proportion));
```

See the Debug Logging by Module Level for more information on the following topics:

- The macros and functions you can call to do debugging logging from code you write.
- How to enable and disable debugging logging by module level at run time.
- How to indicate the destination of the output of the debugging log.

IOStream Sample Code

The C and C++ helpers provided in the IOStream helper library, SampIOS.lib, provide text output of the IBaseFilter interface and other DirectShow objects. The output from these helpers might be useful during debugging, to help understand the details of a given pin or filter. You can use these helpers in your DirectShow filters and applications. For more information about this library, see SampIOS Sample (IOStream Helper Library).

Critical Section Usage

To make deadlocks easier to track, insert assertions in the code that determine whether a critical section is owned by the calling code. The CritCheckIn and CritCheckOut functions indicate whether the calling thread owns the given critical sections, and are generally called in ASSERT macros. For more information about these functions, see CCritSec Debug Functions.

For debug logging of each lock and unlock of a given critical section, you might want to use the DirectShow DbgLockTrace function.

Note Logging can affect performance.

Pointer Validation

Consider using the pointer validation macros. For example, you can call ValidateReadPtr to ensure that the given pointer actually points to readable memory. Note the performance cost of each of these calls. Currently, the DirectShow pointer validation macros are built on top of the Win32 pointer validation functions such as IsBadReadPtr. On some systems, the Win32 pointer validation functions swap in every page in the range specified. For more information about validation macros, see Pointer Validation Macros.

DLL Base Address Conflicts

If you copy any sample makefile to create any new DLL, including filters and plug-in distributors (PIDs), ensure you change the base address to avoid collisions with other DLLs. A *collision* of DLL load address results in one of the DLLs having to be relocated during the time of loading. This increases the load time for that DLL.

In the sample makefiles, the DLL base address is set in DLL_BASE, which is used in ActiveX.mak. Do not let ActiveX.mak use the default value for DLL_BASE, because this will cause collisions.

Using Different Kinds of Builds

DirectShow can be built for three kinds of builds: retail, debug, and performance. See Reserved Identifiers for information on the kinds of builds. Debugging has varying degrees of difficulty for the three kinds of builds, depending on the situation. For instance, the debug build can provide much more information, but it can run so slowly as to make real-time

debugging impossible.

The binaries you create must match the kind of build you're using. The makefiles provided for each sample use ActiveX.mak, which comes with the DirectShow SDK. Comments at the head of ActiveX.mak explain the various nmake command-line parameters to use to obtain binaries compatible with the different DirectShow builds. Some of these parameters define identifiers like DEBUG and PERF when compiling the C or C++ code.

If you must have build-dependent code, you can conditionally compile with the same identifiers that the DirectShow headers use for that purpose. See Reserved Identifiers for a list of the identifiers reserved for that purpose.

For instance, in C or C++, you can conditionally compile code like this:

```
    ...   /* normal processing */
#ifdef DEBUG
    ...   /* debug only code */
#endif
    ...   /* resume normal processing */
```

Debugging New Filters

This section discusses the following points of which you should be aware when debugging new filters:

- Avoid GUID Conflicts
- Test With the Filter Graph Editor and Other Sample Filters
- Add the Filter as an Additional DLL in Developer Studio

Avoid GUID Conflicts

DirectShow uses globally unique identifiers (GUIDs) to find each filter, pin, and property page. Avoid reusing any of the same **GUID**s when copying from the DirectShow sample code. The Guidgen.exe and Uuidgen.exe utilities generate unique **GUID**s.

Test With the Filter Graph Editor and Other Sample Filters

Register your new filter. See Register DirectShow Objects and AMovieDllRegisterServer2 for information about registering a filter.

After you have registered your filter, you can use a tool called the Filter Graph Editor (also called GraphEdit, or Graphedt.exe) to insert your filter into a filter graph and connect it to other filters. You can access GraphEdit from the DXMedia SDK program group. Run GraphEdit and choose Insert Filters from the Graph menu to insert your filter.

If you are debugging an audio filter, there are two sample filters you might consider connecting to your filter to make sure it behaves as expected. You can also look at the source code for those samples to see how they implement methods and member functions. For overviews of those code samples, see Synth Sample (Audio Synthesizer Filter) and Scope Sample (Oscilloscope Filter).

After you have the Filter Graph Editor successfully loading your new filter, you can use the File Dump Filter (Dump.ax) as a useful debugging tool. For instance, it can be used to verify, bit by

bit, the results of a transform filter. Build a graph manually using the Filter Graph Editor and hook the File Dump Filter onto the output of a transform or any other pin. You can also hook up the Inftee Sample (Infinite-Pin Tee Filter) (InfTee.ax), and put the File Dump Filter on one leg of the tee and the "normal" output on another to monitor what happens in the real-time case. For more information, see Dump Sample (Dump Filter).

Add the Filter as an Additional DLL in Developer Studio

If you're going to debug your filter with Microsoft Developer Studio version 5.0, you must tell the debugger about your filter. Here are the steps you should follow in Developer Studio to identify your filter as being a debuggable DLL:

1. From the Project menu, choose Settings....
2. Select the Debug tab.
3. Choose "Additional DLLs" from the Category drop-down list.
4. Add "myfilter.ax" to the list, where "myfilter" is the name of your filter.

Detecting Leaks

Detecting and fixing memory leaks is another important debugging topic.

Visual C++ has an optional debug heap, which can be useful in tracking down memory leaks. (See the "Using the Debug Heap" section of the Visual C++ documentation for more information.) For example, the Visual C++ _CrtSetDbgFlag function enables you to turn on the memory-leak-checking flag bit.

Other providers of memory leak tools can be found in the Microsoft Enterprise Development Partners directory.

Another kind of leak is of COM object references. You can track down object reference leaks by performing the following steps.

1. Put a break point on the NonDelegatingAddRef and NonDelegatingRelease methods of that object.
2. Use Developer Studio (or another debugger) and step through every reference count change, trying to pair them up.
3. Look at the call stack for each change.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

Assert Macros and Functions

The Microsoft® DirectShow™ SDK has three assertion macros: ASSERT, EXECUTE_ASSERT, and KASSERT. The most commonly used assertion macro is **ASSERT**. If **ASSERT** fails,

DirectShow displays a message box that lists the file and line number of the macro call. The **EXECUTE_ASSERT** macro is similar to **ASSERT** except that the condition will still be evaluated in a build of any kind. The third assertion macro is **KASSERT**, which is more suitable for pure filters, such as those in the kernel, where the condition is printed onto the debugger rather than to a message box.

There are also two assertion functions: DbgAssert and DbgKernelAssert. You should call the assertion functions from assertion macros, rather than from normal code.

Name	Description
ASSERT	Checks an assertion in a debug build.
DbgAssert	Handles an assertion failure in a debug build.
DbgKernelAssert	Handles a kernel assertion failure in a debug build.
EXECUTE_ASSERT	Always evaluates a condition; if it is not TRUE in a debug build, treat this as an assertion failure.
KASSERT	Checks a kernel assertion in a debug build.

Previous Home Topic Contents Index Next

ASSERT

Assert Macros and Functions

Evaluates the given condition in a debug build. If the resulting evaluation is false, then **ASSERT** calls DbgAssert to handle the assertion failure. **DbgAssert** can return to its caller later, if the user so desires.

ASSERT(
cond
);

Parameters

cond
Boolean expression that defines the condition to evaluate.

Remarks

If you use the ASSERT macro, DbgAssert might display a message box. If this is not acceptable in your environment, you can use KASSERT (kernel assert) instead.

Here are two examples of **ASSERT** calls:

```
ASSERT( First != NULL);
ASSERT( StartTime <= EndTime);
```

DbgAssert

Assert Macros and Functions

Handles an assertion failure in a debug build. **DbgAssert** will display a message box that includes the condition text, source file name, and source line number. The user will be given the choice to ignore the assertion failure, debug the assertion, or force the application to exit. Thus **DbgAssert** might return to the caller, depending on the user's actions.

```
void WINAPI DbgAssert(
  const TCHAR *pCondition,
  const TCHAR *pFileName,
  INT iLine
  );
```

Parameters

pCondition
Pointer to a string version of a Boolean expression.
pFileName
Pointer to a source file name.
iLine
Line number within the source file.

Remarks

This function is available only in a debug build. Usually, **DbgAssert** will be called by macros such as ASSERT, not directly from other code.

If you use the ASSERT macro, **DbgAssert** might display a message box. If this is not acceptable in your environment, you can use DbgKernelAssert instead.

DbgKernelAssert

Assert Macros and Functions

Called in a debug build to print the condition onto the kernel debugger, including the condition text, source file name, and source line number.

```
void WINAPI DbgKernelAssert(
  const TCHAR *pCondition,
  const TCHAR *pFileName,
  INT iLine
  );
```

Parameters

pCondition
Pointer to a string version of a Boolean expression.
pFileName
Pointer to a source file name.
iLine
Line number within the source file.

Remarks

This function is available only in a debug build. Usually, **DbgKernelAssert** is called by macros such as KASSERT, not directly from other code.

Unlike DbgAssert, when macros call **DbgKernelAssert** in a debug build, no message box appears.

Previous Home Topic Contents Index Next

EXECUTE_ASSERT

Assert Macros and Functions

Evaluates the condition. In a debug build, if the resulting value is not TRUE, then the **EXECUTE_ASSERT** macro will invoke DbgAssert to handle the assertion failure. **DbgAssert** might return to the caller, depending on the user's actions.

```
EXECUTE_ASSERT(
  cond
  );
```

Parameters

cond
Condition (a Boolean expression), which is always evaluated. This contrasts with ASSERT and many other traditional assertion macros, which do not evaluate the condition in

nondebug builds.

Remarks

If you use the EXECUTE_ASSERT macro in a debug build, DbgAssert might display a message box. If this is not acceptable in your environment, you can use KASSERT (kernel assert) instead.

Previous Home Topic Contents Index Next

KASSERT

Assert Macros and Functions

In a debug build, if the condition evaluates to FALSE, the **KASSERT** macro prints the condition on the kernel debugger, including the file name and line number.

KASSERT(
cond
);

Parameters

cond
Condition (a Boolean expression).

Remarks

This macro is ignored unless DEBUG is defined when the Microsoft DirectShow headers are included.

Unlike ASSERT and EXECUTE_ASSERT, if you use this macro in a debug build no message box will appear.

Previous Home Topic Contents Index Next

Breakpoint Macros and Function

You can use the breakpoint macros and function to break into the debugger (either the regular debugger or the kernel debugger). For example, DbgBreak causes a regular debugger

breakpoint, whereas KDbgBreak causes a kernel debugger breakpoint.

Name	Description
DbgBreak	Breakpoint with message box.
DbgBreakPoint	Breakpoint with message box.
KDbgBreak	Breakpoint with message on kernel debugger.

Previous Home Topic Contents Index Next

DbgBreak

Breakpoint Macros and Function

Generates a message box in a debug build with the indicated string literal, the source file name, and the source line number. Buttons in the message box enable you to break into the debugger, kill the application, or ignore the message box.

DbgBreak(
strLiteral
);

Parameters

strLiteral
Text string, which must be in quotation marks.

Remarks

This macro is ignored unless DEBUG is defined when the Microsoft® DirectShow™ headers are included.

Previous Home Topic Contents Index Next

DbgBreakPoint

Breakpoint Macros and Function

Generates a message box in a debug build with the indicated string literal, the source file name, and the source line number. Buttons in the message box enable you to break into the debugger, kill the application, or ignore the message box.

```
void WINAPI DbgBreakPoint(
  const TCHAR *pCondition,
  const TCHAR *pFileName,
  INT iLine
  );
```

Parameters

pCondition
Pointer to a string indicating what happened.
pFileName
Pointer to a source file name.
iLine
Line number within the source file.

Remarks

This function is available only in a debug build.

Previous Home Topic Contents Index Next

KDbgBreak

Breakpoint Macros and Function

Generates a kernel debugger message in a debug build with the indicated string literal, the source file name, and the source line number.

```
KDbgBreak(
  strLiteral
  );
```

Parameters

strLiteral
Text string, which must be in quotation marks.

Remarks

This macro is ignored unless DEBUG is defined when the Microsoft DirectShow headers are included.

Previous Home Topic Contents Index Next

Debug Output

The debug output facility is just one of several facilities of debug logging available with Microsoft® DirectShow™. This facility is monolithic (that is, it is either on or off). For information about the facility with the most precise control enabling and disabling logging, see Debug Logging by Module Level. For information about the simplest facility, see Debug NOTE (Message) Macros.

For more information about how DbgOutString chooses the debug output location, see Debug Output Location. DbgInitialise opens the debug output location and DbgTerminate closes it.

Name	Description
DbgOutString	Sends a debug string to the debug output location.
DumpGraph	Sends debugging information from the filter graph to the debug output location.

Previous Home Topic Contents Index Next

DbgOutString

Debug Output

Outputs the given string to the debug output location.

void WINAPI DbgOutString(
LPCTSTR *psz*
);

Parameters

psz
Pointer to a string to be output.

Remarks

DbgOutString is ignored unless DEBUG is defined when the Microsoft DirectShow headers are included. That is, it is a function in a debug build, and a macro that does nothing in other builds.

Previous Home Topic Contents Index Next

DumpGraph

Debug Output

Sends debugging information from the filter graph to the debug output location.

void WINAPI (
IFilterGraph **pGraph*,
DWORD *dwLevel*
)

Parameters

pGraph
Pointer to the filter graph to get debugging information about.
dwLevel
Logging level for this message, where zero means always log.

Remarks

Call this helper function to send potential error messages after you instantiate a filter graph.

Previous Home Topic Contents Index Next

Debug NOTE (Message) Macros

The NOTE macros provide debug message abilities in the Microsoft® Foundation Class Library (MFC) style. For more information about how the **NOTE** macros are implemented, see Debug Logging by Module Level. The **NOTE** macros work like a call to DbgLog, with a message type of LOG_TRACE, and a logging level of 5. For more information about how **NOTE** macros choose the debug output location, see Debug Output Location.

Macro	Description
NOTE	Logs a debug message with zero additional parameters.
NOTE1	Logs a debug message with one additional parameter.
NOTE2	Logs a debug message with two additional parameters.
NOTE3	Logs a debug message with three additional parameters.
NOTE4	Logs a debug message with four additional parameters.
NOTE5	Logs a debug message with five additional parameters.

Previous Home Topic Contents Index Next

NOTE through NOTE5 Macros

Will format and print their parameters on the debugger.

NOTE(
pFormat
);
NOTEx(
pFormat,
[a-e]
);

Parameters

pFormat
A printf-style format string, which must be in quotation marks.
a through *e*
Optional parameters, each of which must have a respective format string portion (such as `"%d"`).

Remarks

These macros are ignored unless DEBUG is defined when the Microsoft DirectShow™ headers are included. The following example shows the syntax for the **NOTE1** through **NOTE5** macros.

```
NOTE1(pFormat, a);
NOTE2(pFormat, a, b);
NOTE3(pFormat, a, b, c);
NOTE4(pFormat, a, b, c, d);
NOTE5(pFormat, a, b, c, d, e);
```


Previous Home Topic Contents Index Next

Pointer Validation Macros

Microsoft® DirectShow™ provides some macros to make pointer usage more robust. These include a simple CheckPointer macro (which tests if a given pointer is NULL). These also include a number of **ValidateXxxPtr** macros, which ensure a given pointer actually refers to