The default implementation returns m_bEOSDelivered. This is used by the base renderer class so that only one EC_COMPLETE message is sent to the filter graph manager each time it is run, regardless of the number of times EndOfStream is called.

Previous Home Topic Contents Index Next

CBaseRenderer::IsStreaming

CBaseRenderer Class

Determines if the filter is streaming data.

BOOL IsStreaming(void);

Return Values

Returns TRUE if the renderer is rendering, or FALSE if it isn't.

Remarks

The default implementation returns m_bStreaming. In the base renderer class, "streaming" and "rendering" are used in the same context as "running".

Previous Home Topic Contents Index Next

CBaseRenderer::NonDelegatingQueryInterface

CBaseRenderer Class

Retrieves 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 overrides CBaseFilter::NonDelegatingQueryInterface. It exposes the IMediaPosition and IMediaSeeking interfaces and then calls **CBaseFilter::NonDelegatingQueryInterface** for interfaces implemented in the base classes.

CBaseRenderer::NotReady

CBaseRenderer Class

Forces the m_evComplete event into a nonsignaled state.

void NotReady(void);

Return Values

No return value.

Remarks

This member function calls the CAMEvent::Reset member function of the m_evComplete event object.

CBaseRenderer::NotifyEndOfStream

CBaseRenderer Class

Sends an EC_COMPLETE event to the filter graph manager.

void NotifyEndOfStream(void);

Return Values

No return value.

CBaseRenderer::OnReceiveFirstSample

CBaseRenderer Class

Provides derived classes with an opportunity to render static data.

virtual void OnReceiveFirstSample(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
    Media sample.

Return Values

No return value.

Remarks

This member function is unimplemented. It is primarily used by video renderers. When they receive their first sample while paused, they typically draw the frame as a poster image. This virtual method is called by the base classes when the first sample arrives.

CBaseRenderer::OnRenderEnd

CBaseRenderer Class

Notifies the derived class that rendering has finished.

virtual void OnRenderEnd(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
Media sample.

Return Values

No return value.

Remarks

This member function is available for quality management and performance measuring. It is called immediately after the sample is rendered.

Quality management implementations typically need to know how long it takes the renderer to render the data.

CBaseRenderer::OnRenderStart

CBaseRenderer Class

Notifies the derived class that rendering is about to start.

virtual void OnRenderStart(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
Media sample.

Return Values

No return value.

Remarks

This member function is available for quality management and performance measuring. It is called immediately before the sample is rendered.

Quality management implementations typically need to know how long it takes the renderer to render the data.

CBaseRenderer::OnStartStreaming

CBaseRenderer Class

Notifies the derived class that streaming has started.

virtual HRESULT OnStartStreaming(void);

Return Values

Returns NOERROR in the default implementation.

Remarks

This member function is called from CBaseRenderer::StartStreaming. Override this in your derived class to provide special handling when streaming starts.

CBaseRenderer::OnStopStreaming

CBaseRenderer Class

Notifies the derived class that streaming has stopped.

virtual HRESULT OnStopStreaming(void);

Return Values

Returns NOERROR in the default implementation.

Remarks

This member function is called from CBaseRenderer::StopStreaming. Override this in your derived class to provide special handling when streaming stops.

Previous Home Topic Contents Index Next

CBaseRenderer::OnWaitEnd

CBaseRenderer Class

Notifies the derived class that a wait for a rendering time has just ended.

virtual void OnWaitEnd(void);

Return Values

No return value.

Remarks

This member function is available for quality control and is called from CBaseRenderer::WaitForRenderTime just after waiting for the presentation time for a sample. Override this member function to obtain performance measurements in a derived class.

Previous Home Topic Contents Index Next

CBaseRenderer::OnWaitStart

CBaseRenderer Class

Notifies the derived class that a wait for a rendering time is about to start.

virtual void OnWaitStart(void);

Return Values

No return value.

Remarks

This member function is available for quality control and is called from CBaseRenderer::WaitForRenderTime just before waiting for the presentation time for a sample. Override this member function to obtain performance measurements in a derived class.

CBaseRenderer::Pause

CBaseRenderer Class

Changes the renderer to State_Paused if it isn't already.

HRESULT Pause(void);

Return Values

Returns an HRESULT value.

Remarks

The following steps comprise a pause operation.

1. Commit the allocator used for the connection.
2. Allow the thread for the upstream filter to wait in Receive.
3. Cancel any outstanding clock advise links.
4. Check to see if the renderer is connected and allow a state change.
5. If a sample is available, complete the state change to State_Paused.

If the member function succeeds, DirectShow sets the filter's m_State member variable to State_Paused. If the renderer is in the State_Stopped state, DirectShow calls the CBasePin::Active member function for each of the renderer's connected pins.

This member function overrides CBaseFilter::Pause.

CBaseRenderer::PrepareReceive

CBaseRenderer Class

Ensures that a sample can be rendered.

virtual HRESULT PrepareReceive(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
    Media sample.

Return Values

Returns NOERROR if successful, VFW_E_SAMPLE_REJECTED if the delivered sample is later than the sample's timestamp, or E_UNEXPECTED if a renderable sample is already available.

Remarks

This member function is called when the upstream filter delivers a sample. If the upstream filter is running (streaming), the sample is scheduled with the reference clock. If the upstream filter is not streaming, a sample in paused mode has been received, so any state transition can be completed. On leaving this function, everything will be unlocked so an application thread can get in and change the state to stopped. In this case, it will also signal the thread event so that the wait call is stopped.

This function is typically called from the IMemInputPin::Receive method on the renderer's input pin. Although **PrepareReceive** returns VFW_E_SAMPLE_REJECTED if the sample was delivered too late to be useful, the **IMemInputPin::Receive** method should not pass the VFW_E_SAMPLE_REJECTED error on to the upstream filter in this case. Instead, **IMemInputPin::Receive** should return NOERROR, because no error occurred.

CBaseRenderer::PrepareRender

CBaseRenderer Class

Provides an opportunity for the derived class to prepare itself for rendering a sample.

virtual void PrepareRender(void);

Return Values

No return value.

Remarks

This member function is called from CBaseRenderer::Receive before rendering each frame. A derived class can take this opportunity to prepare itself for rendering. For example, a video renderer might realize its palette. This is not implemented in the base class.

Previous Home Topic Contents Index Next

CBaseRenderer::Ready

CBaseRenderer Class

Puts the m_evComplete event into a signaled state.

void Ready(void);

Return Values

No return value.

Remarks

This member function calls the m_evComplete CAMEvent object's Set member function.

Previous Home Topic Contents Index Next

CBaseRenderer::Receive

CBaseRenderer Class

Called by the upstream filter when a sample is available to render.

virtual HRESULT Receive(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
Media sample.

Return Values

Returns an HRESULT value.

Remarks

This member function sets an advise link with the clock, waits for the time to arrive, and then renders the data by calling the pure virtual DoRenderSample member function that the derived class will have overridden. After rendering the sample, the end of stream can also be signaled if it was the last one sent before EndOfStream was called.

Previous Home Topic Contents Index Next

CBaseRenderer::Render

CBaseRenderer Class

Asks the derived class to render the sample.

virtual HRESULT Render(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
Media sample.

Return Values

Returns an HRESULT value.

Remarks

This member function is called when the derived class should render the sample. The action taken is dependent on the nature of the renderer; a video renderer will typically draw the image in a window. This class calls the pure virtual DoRenderSample to be implemented by the derived class.

Previous Home Topic Contents Index Next

CBaseRenderer::ResetEndOfStream

CBaseRenderer Class

Resets the end-of-stream flag.

virtual HRESULT ResetEndOfStream(void);

Return Values

Returns an HRESULT value.

Remarks

This member function is typically called when changing to stopped states. A renderer must keep track of when it gets told that no more data is going to arrive (this is done when the sourcing filter calls IPin::EndOfStream). At this point the renderer finishes rendering any data it has and then sends an EC_COMPLETE event to the filter graph manager.

However, when the filter is stopped, the whole state is cleared. When the filter is subsequently run, the source filter will signal the end of stream again if it has no data to send. In this case, the renderer should signal another EC_COMPLETE event to the filter graph manager. This member function resets the state so that when next requested it will send an EC_COMPLETE event.

Previous Home Topic Contents Index Next

CBaseRenderer::ResetEndOfStreamTimer

CBaseRenderer Class

If the end-of-stream timer is nonzero, this function sets it to zero.

void ResetEndOfStreamTimer(void);

Return Values

No return value.

CBaseRenderer::Run

CBaseRenderer Class

Transitions the renderer to State_Running if it is not in this state already.

HRESULT Run(void);

Return Values

Returns an HRESULT value.

Remarks

If the renderer is in the State_Stopped state, the CBaseRenderer::Pause member function is called first to transition the renderer to the State_Paused state, which has the effect of activating any of the filter's connected pins. If this member function succeeds, the renderer's m_State member variable is set to State_Running.

This member function overrides CBaseFilter::Run.

CBaseRenderer::ScheduleSample

CBaseRenderer Class

Schedules the sample for rendering.

virtual BOOL ScheduleSample(
 IMediaSample **pMediaSample*
 );

Parameters

pMediaSample
 Media sample.

Return Values

No return value.

Remarks

One of the main purposes of the renderer base class is to manage the timing and synchronization of the samples it is sent; that is, the timely presentation of data. It also must look after quality management, which might involve dropping samples or rendering them earlier than indicated in the time stamps on the sample. This method and its overrides in derived classes manage the setting up of advise links with the clock, so that the samples can be rendered at the appropriate time.

Previous Home Topic Contents Index Next

CBaseRenderer::SendEndOfStream

CBaseRenderer Class

Signals an EC_COMPLETE event to the filter graph manager.

virtual HRESULT SendEndOfStream(void);

Return Values

Returns an HRESULT value.

Remarks

When the renderer receives an end-of-stream notification, it will finish rendering any data it currently has and then send an EC_COMPLETE event to the filter graph manager.

Previous Home Topic Contents Index Next

CBaseRenderer::SendNotifyWindow

CBaseRenderer Class

Passes the notification window handle to the upstream filter.

void SendNotifyWindow(
  IPin **pPin*,
  HWND *hwnd*
  );

Parameters

pPin
    IPin interface of the upstream pin.
hwnd
    Handle of the notification window.

Return Values

No return value.

Remarks

If the output pin of the upstream filter supports the IMediaEventSink interface, this member function sends it the EC_NOTIFY_WINDOW event code with the window handle in *hwnd*.

Previous Home Topic Contents Index Next

CBaseRenderer::SendRepaint

CBaseRenderer Class

Signals an EC_REPAINT message to the filter graph.

void SendRepaint(void);

Return Values

No return value.

Remarks

This should be used with some care. EC_REPAINT events are processed by the filter graph manager by setting the current position to the same position that the graph is currently in. This has the effect of sending the same data through the graph again, which is an expensive operation. Video renderers are the main users of this event, because they sometimes need the same image sent again to refresh the display.

CBaseRenderer::SetAbortSignal

CBaseRenderer Class

Sets the m_bAbort abort signal flag.

void SetAbortSignal(
  BOOL *bAbort*
  );

Parameters

bAbort
    Abort value to be set.

Return Values

Returns an HRESULT value.

CBaseRenderer::SetMediaType

CBaseRenderer Class

Informs the derived class of the selected media type.

virtual HRESULT SetMediaType(

const CMediaType **pmt*
);

Parameters

pmt
Media type to be set.

Return Values

Returns NOERROR by default; the overriding member function should return a valid HRESULT value.

Remarks

This member function is called by the CRendererInputPin::SetMediaType member function and has no implementation in this class. Derived classes can optionally override to add functionality.

Previous Home Topic Contents Index Next

CBaseRenderer::SetRepaintStatus

CBaseRenderer Class

Resets the m_bRepaintStatus flag when EC_REPAINT has been signaled to the filter graph.

void SetRepaintStatus(
BOOL *bRepaint*
);

Parameters

bRepaint
Boolean value assigned to the m_bRepaintStatus flag.

Return Values

No return value.

Remarks

The m_bRepaintStatus flag ensures that the filter graph is not flooded with redundant calls. Once one EC_REPAINT message has been sent, no more will be sent until the renderer receives some data.

Previous Home Topic Contents Index Next

CBaseRenderer::ShouldDrawSampleNow

CBaseRenderer Class

Determines if the sample should be drawn between the start and stop times given.

virtual HRESULT ShouldDrawSampleNow(
  IMediaSample **pMediaSample*,
  REFERENCE_TIME **pStartTime*,
  REFERENCE_TIME **pEndTime*
  );

Parameters

pMediaSample
  Media sample.
pStartTime
  Start time in question.
pEndTime
  End time in question.

Return Values

Returns S_FALSE by default. The overriding member function can return S_OK to indicate that the sample should be drawn immediately instead of waiting for its scheduled time.

Remarks

This member function is used by the derived video renderer class for quality management.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseRenderer::SignalTimerFired

CBaseRenderer Class

Resets the current advise time to zero after a timer fires.

virtual void SignalTimerFired(void);

Return Values

No return value.

CBaseRenderer::SourceThreadCanWait

CBaseRenderer Class

Sets or resets the thread event.

virtual HRESULT SourceThreadCanWait(
  BOOL *bCanWait*
  );

Parameters

bCanWait
  TRUE or FALSE, depending on intent.

Return Values

Returns an HRESULT value.

Remarks

In some states, such as paused or running, it is expected that the upstream filter's thread will be blocked in the call to the renderer's input pin Receive method. In other cases, such as when the renderer is stopped, the upstream filter should not be required to wait. This member function represents a manual reset event that sets this TRUE to wait, or FALSE to keep the thread from waiting.

CBaseRenderer::StartStreaming

CBaseRenderer Class

Called to schedule any pending sample with the clock, and to display timing information.

virtual HRESULT StartStreaming(void);

Return Values

Returns an HRESULT value.

Remarks

If no sample is available but an end-of-stream flag is queued, this member function sends an EC_COMPLETE message to the filter graph manager. If a sample is available, the EC_COMPLETE message will not be sent until it has been rendered.

Previous Home Topic Contents Index Next

CBaseRenderer::Stop

CBaseRenderer Class

Transitions the renderer to State_Stopped if it is not in this state already.

HRESULT Stop(void);

Return Values

Returns an HRESULT value.

Remarks

If the renderer is not in the State_Stopped state, the CRendererInputPin::Inactive member function is called for each of the renderer's connected pins. If this member function succeeds, the filter's m_State member variable is set to State_Stopped.

This member function overrides CBaseFilter::Stop.

Previous Home Topic Contents Index Next

CBaseRenderer::StopStreaming

CBaseRenderer Class

Sets the internal flag to indicate not to schedule arrival of any more samples.

virtual HRESULT StopStreaming(void);

Return Values

Returns an HRESULT value.

Remarks

Call this member function when streaming stops. The state change methods in the filter implementation take care of canceling any clock advise link that has been set up and clearing any pending sample.

Previous Home Topic Contents Index Next

CBaseRenderer::TimerCallback

CBaseRenderer Class

Checks if it is time to signal the end of the current data stream.

void TimerCallback(void);

Return Values

No return value.

Remarks

If the m_EndOfStreamTimer data member is nonzero, this function sets it to zero and calls CBaseRenderer::SendEndOfStream to signal the end of the current data stream.

Previous Home Topic Contents Index Next

CBaseRenderer::WaitForReceiveToComplete

CBaseRenderer Class

Waits for the CBaseRenderer::Receive method to complete.

void WaitForReceiveToComplete( );

Return Values

No return value.

Remarks

Use this method when you wish to avoid deadlock which occurs when CBaseRenderer::Stop is called and the CBaseRenderer::Receive hasn't completed.

CBaseRenderer::WaitForRenderTime

CBaseRenderer Class

Waits for either the due time for the current sample to arrive or for rendering to be stopped.

virtual HRESULT WaitForRenderTime(void);

Return Values

Returns an HRESULT value.

Remarks

The member function is virtual because derived classes might have more events that they also want to wait on, which might interrupt the waiting process. The base class has two events: m_RenderEvent and m_ThreadSignal. The former is signaled by the clock when the sample is due for rendering. The latter is signaled by the filter when it should give up waiting and abort (making the assumption that the filter was stopped).

CBaseStreamControl Class

IAMStreamControl
CBaseStreamControl

The **CBaseStreamControl** class implements the IAMStreamControl interface on input and output pins in a filter graph. This class provides control of the starting and stopping of various components of the stream. Various streams can be turned on or off without affecting the rest of the graph. For example, an audio stream can be turned off while a video stream continues, for muting. Or perhaps a capture stream can be turned off while preview continues to flow. This could be used to assist in frame accuracy when exact capture start or stop times are important.

CBaseStreamControl enables you to specify start and stop times in the StartAt and StopAt member functions and provides stream information in the GetInfo member function. **CBaseStreamControl** uses the StreamControlState enumerated data type to describe the various states a stream is in. If a stream is flowing it is indicated by the STREAM_FLOWING setting, otherwise it is in a discarding state indicated by the STREAM_DISCARDING setting.

Filters that need to implement the interface on their own should typically inherit from **CBaseStreamControl** to obtain an implementation of the StartAt, StopAt, and GetInfo methods. The **CBaseStreamControl** class also maintains state information and decides what to do with the sample. To implement your own filter with pins that support **CBaseStreamControl** you must:

- Inform the filter object of all state changes through the NotifyFilterState member function.
- Inform the filter object of all SetSyncSource calls to the filter.
- Inform the filter object when in a flushing state, and when flushing has completed, in the CBaseStreamControl::Flushing member function.
- Use the CheckStreamState function to make decisions about discarding or passing along samples.
- Make sure output pins set discontinuity flags on the first sample flowed after samples have been discarded.
- Tell your pin what the sink is when your filter joins a filter graph, as shown in the following example.

```
STDMETHODIMP CMyFilter::JoinFilterGraph(IFilterGraph * pGraph, LPCWSTR pName)
{
   HRESULT hr = CBaseFilter::JoinFilterGraph(pGraph, pName);
   if (hr == S_OK)
      m_pMyPin->SetFilterGraph(m_pSink);
   return hr;
}
```

If you are implementing the IAMStreamControl interface without using **CBaseStreamControl**, the last two preceding points do not apply.

For sample code see the video capture sample at DXmedia\Samples\DS\vidcap.

Member Functions

Name	**Description**
CBaseStreamControl	Constructs a CBaseStreamControl object.
CheckStreamState	Retrieves a stream's current state.
Flushing	Notifies the pin when the filter is flushing.
GetInfo	Retrieves information about the current streaming settings.
NotifyFilterState	Notifies the pin of what state your filter is in.
SetFilterGraph	Sets the event sink notification that your filter graph is using.
SetSyncSource	Identifies the reference clock being used by the graph your filter is in.
StartAt	Informs the pin when to start sending streaming data.
StopAt	Informs the pin when to stop processing data and discard any new samples.

CBaseStreamControl::CBaseStreamControl

CBaseStreamControl Class

Constructs a CBaseStreamControl object.

CBaseStreamControl( );

Return Values

No return value.

Remarks

This method initializes start time and stop time to MAX_TIME, which implies that times are unspecified.

CBaseStreamControl::CheckStreamState

CBaseStreamControl Class

Retrieves a stream's current state.

```
enum StreamControlState CheckStreamState( IMediaSample * pSample );
```

Values

pSample
 Pointer to an IMediaSample interface.

Return Values

Returns a StreamControlState enumeration type.

Remarks

Your filter calls this member function when your pin receives a sample that it is about to forward. The first sample you forward after throwing one or more away should be marked as a discontinuity.

If your filter implements the IAMDroppedFrames interface and is counting how many frames are dropped, it should not count a frame that is discarded as dropped.

The following example shows what you should include if your filter inherits from CBaseStreamControl.

```
//Pin has been given a sample to pass on, pSample
//m_fLastSampleDiscarded is initialized to TRUE when streaming starts

int iStreamState = CheckStreamState(pSample);
if (iStreamState == STREAM_FLOWING) {
   if (m_fLastSampleDiscarded)
      pSample->SetDiscontinuity(TRUE);
   m_fLastSampleDiscarded = FALSE;
   //now deliver it or put it o a queue to be delivered, or whatever.
} else {
   m_fLastSampleDiscarded = TRUE;     //next one is discontinuity
   //do NOT deliver this sample. Just throw it away
}
```

Previous Home Topic Contents Index Next

CBaseStreamControl::Flushing

CBaseStreamControl Class

Notifies the pin that the filter is flushing.

void Flushing(
  BOOL *bInProgress* **);**

Parameters

bInProgress
    TRUE indicates flushing in progress; FALSE indicates not flushing.

Return Values

No return value.

Remarks

If you are implementing your own filter, your pin must call this member function on BeginFlush and EndFlush (DeliverBeginFlush and DeliverEndFlush for output pins) to say when it is flushing, as shown in the following example.

```
HRESULT CMyPin::BeginFlush()
{
   Flushing(TRUE);
   //or CBaseInputPin for input pins
   return CBaseOutputPin::BeginFlush();
}

HRESULT CMyPin::EndFlush()
{
   Flushing(FALSE);
   //or CBaseInputPin for input pins
   return CBaseOutputPin::EndFlush();
}
```

Note that capture filters that do not support seeking do not call this method.

Previous Home Topic Contents Index Next

CBaseStreamControl::GetInfo

CBaseStreamControl Class

Retrieves information about the current streaming settings.

HRESULT GetInfo(
  AM_STREAM_INFO **pInfo*
  );

Parameters

pInfo
  Pointer to an AM_STREAM_INFO structure.

Return Values

Returns S_OK.

Remarks

This member function implements the IAMStreamControl interface and is called by the user to find out if a pin is streaming and to obtain the stream's attributes.

Previous Home Topic Contents Index Next

CBaseStreamControl::NotifyFilterState

CBaseStreamControl Class

Notifies the pin of your filter's state.

void NotifyFilterState(
  FILTER_STATE *new_state***,**
  REFERENCE_TIME *tStart* **= 0);**

Parameters

new_state
  Filter's new state.
tStart
  Time at which streaming starts (only valid when *new_state* is in *State_Running*).

Return Values

No return value.

Remarks

This member function notifies the pin of a filter's new state by setting a FILTER_STATE enumeration type variable.

If you are implementing your own filter, inform your pin's **CBaseStreamControl::NotifyFilterState** member function what state your filter is in every time your filter changes state, as shown in the following example.

```
STDMETHODIMP CMyFilter::Run(REFERENCE_TIME tStart)
{
   //once error check is successful
   m_pMyPin->NotifyFilterState(State_Running, tStart);

   //now continue with whatever should occur next, for example...
   return CBaseFilter::Run(tStart);
}

STDMETHODIMP CMyFilter::Pause()
{
   //once error check is successful
   m_pMyPin->NotifyFilterState(State_Paused, 0);

   //now continue with whatever should occur next, for example...
   return CBaseFilter::Pause();
}

STDMETHODIMP CMyFilter::Stop()
{
   //once error check is successful
   m_pMyPin->NotifyFilterState(State_Stopped, 0);

   //now continue with whatever should occur next, for example...
   return CBaseFilter::Stop(tStart);
}
```

CBaseStreamControl::SetFilterGraph

CBaseStreamControl Class

Sets the event sink notification your filter graph is using.

void SetFilterGraph(
  IMediaEventSink **pSink* **)**

Parameters

pSink
  Pointer to an IMediaEventSink interface.

Return Values

No return value.

Remarks

A filter calls this member function in its JoinFilterGraph member function after it creates the IMediaEventSink.

Previous Home Topic Contents Index Next

CBaseStreamControl::SetSyncSource

CBaseStreamControl Class

Identifies the reference clock being used by the graph your filter is in.

void SetSyncSource(
 IReferenceClock * *pRefClock* **);**

Parameters

pRefClock
    Pointer to the IReferenceClock interface.

Return Values

No return value.

Remarks

Filters with pins that use this class should ensure that they pass sync source information to this member function, as shown in the following example.

```
STDMETHODIMP CMyFilter::SetSyncSource(IReferenceClock *pClock)
{
   m_pMyPin->SetSyncSource(pClock);
   return CBaseFilter::SetSyncSource(pClock);
}
```

Previous Home Topic Contents Index Next

CBaseStreamControl::StartAt

CBaseStreamControl Class

Tells the pin when to start sending streaming data.

HRESULT StartAt(
  const REFERENCE_TIME * *ptStart* **= NULL,**
  DWORD *dwCookie* **= 0);**

Parameters

ptStart
  REFERENCE_TIME at which to start streaming. If NULL, start immediately (no notification). If MAX_TIME, start canceled or will have no effect.
dwCookie
  Specifies a particular value, other than 0, to be sent with the notification when the start occurs. (Only used if *ptStart* is non-NULL or MAX_TIME).

Return Values

Returns NOERROR.

Remarks

Streams are enabled by default, so this member function will have no effect unless you have previously called StopAt.

After the stream is in a STREAM_FLOWING state, the filter will send an EC_STREAM_CONTROL_STARTED event notification to the filter graph manager.

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

Previous Home Topic Contents Index Next

CBaseStreamControl::StopAt

CBaseStreamControl Class

Informs the pin when to stop processing data and to discard any new samples.

HRESULT StopAt(
  const REFERENCE_TIME * *ptStop* **= NULL,**
  BOOL *bSendExtra* **= FALSE,**
  DWORD *dwCookie* **= 0);**

Parameters

ptStop
   REFERENCE_TIME at which to stop streaming. If NULL, stop immediately (no notification). If MAX_TIME, cancels stop.

bSendExtra
   Indicates whether to send an extra sample after scheduled *ptStop* time.

dwCookie
   Specifies a particular value to be sent with the notification when the stop occurs. (Only used if *ptStart* if not NULL or MAX_TIME).

Return Values

Returns NOERROR.

Remarks

This member function implements the IAMStreamControl::StopAt method and is used by pins and filters that must support the stopping of streams. It sets the StreamControlState enumeration type to STREAM_DISCARDING.

In a video capture scenario, specify StopAt on both the output pin of a capture filter and an input pin of a multiplexer and have the multiplexer send notification of completion. This ensures that the capture filter doesn't needlessly capture extra frames, while also guaranteeing that the multiplexer has written the last frame to disk.

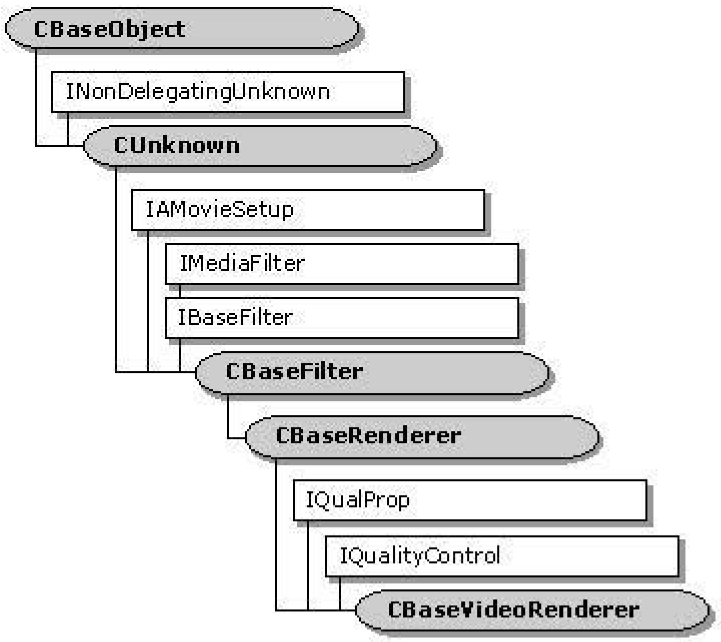
In addition, the capture output pin should specify TRUE for the *bSendExtra* variable while all other pins specify FALSE. If an extra frame is not sent the multiplexer will end up waiting for the stop time indefinitely and not realize it already has received all the capture information.

If you are using ICaptureGraphBuilder, the ICaptureGraphBuilder::ControlStream method will accomplish all this for you automatically.

Note If a stop time is given in the middle of a packet, the filter will deliver the whole packet before going into a discarding state. Also, if start and stop are scheduled for a single point in time, the effect is as if the start occurred an infinitesimal time before the stop. You can use this effect to capture a single frame.

Previous Home Topic Contents Index Next

CBaseVideoRenderer Class



This base class is used for building video renderer filters.

Protected Data Members

Name	Description
m_bDrawLateFrames	Flag to signal that no frames are to be dropped. Debug only. This destroys synchronization.
m_bSupplierHandlingQuality	TRUE indicates quality control messages are being handled. This lets the renderer know to wait until as late as possible to drop frames itself, and to display the next frame very early after the supplier has dropped a frame.
m_cFramesDrawn	Total number of frames that have been drawn since streaming started.
m_cFramesDropped	Cumulative frames that have been dropped in the renderer since streaming started. Frames can also be dropped upstream without the renderer recognizing them.
m_idDecision	MSR_id for the decision code of ShouldDrawSampleNow.
m_idDuration	MSR_id for the duration of a frame.
m_idFrameAccuracy	Performance log identifier for the time in milliseconds that the frame was late.
m_idFrameAvg	Performance log identifier for the average frame time that is used for synchronization and quality control.
m_idQualityRate	MSR_id for the quality rate requested.
m_idQualityTime	MSR_id for the quality time requested.

m_idRenderAvg	Performance log identifier for the average renderer time recorded.
m_idSchLateTime	MSR_id for how late the frame was when scheduled.
m_idSendQuality	MSR_id for timing the notifications (unused).
m_idTimeStamp	MSR_id for a frame time stamp.
m_idWait	Performance log identifier for the recorded wait time (unused).
m_idWaitReal	Performance log identifier for the true wait time.
m_iSumFrameTime	Sum of the interframe times; needed for the property page.
m_iSumSqAcc	Sum of the squares of the accuracies (in milliseconds) needed for the property page.
m_iSumSqFrameTime	Sum of the squares of interframe times; needed for the property page.
m_iTotAcc	Sum of the accuracies (in milliseconds) needed for the property page.
m_nNormal	Number of consecutive frames drawn at their scheduled time. A negative number indicates that a frame has just been dropped by the renderer.
m_trDuration	Duration of the last frame (difference between the start and end times).
m_trEarliness	How early a frame is allowed to be played when a frame has just been dropped.
m_trFrame	Most recently recorded time between frames. Used in statistical measurements.
m_trFrameAvg	Average interframe time in reference time units.
m_trLastDraw	Time of previous frame. Used for interframe time references.
m_trLate	Amount of time that the current frame was late. Used in statistical measurements.
m_trRenderAvg	Time that frames are taking to perform the bit-block transfer.
m_trRenderLast	Time for the last frame bit-block transfer.
m_trRenderStart	Time the bit-block transfer started. Used to get m_trRenderLast.
m_trThrottle	Period to insert after rendering each frame, typically used when audio quality has been increased and video performance must be decreased to allow this.
m_trWaitAvg	Average wait time in reference time units.
m_tStreamingStart	Used for property page statistics. Represents the start time of the current streaming process or the previous streaming process if not currently streaming.

Member Functions

Name	**Description**
CBaseVideoRenderer	Constructs a CBaseVideoRenderer object.
GetStdDev	Estimates the standard deviation in milliseconds between when each frame is due and when it is actually rendered, for per-frame statistics.
PreparePerformanceData	Sets the m_trLate and m_trFrame values of the current frame.
ThrottleWait	Inserts a wait period after each frame.

Overridable Member Functions

Name	Description
JoinFilterGraph	Sends EC_WINDOW_DESTROYED event notification when filter is removed from the filter graph.
OnDirectRender	Collects timing information that controls synchronization and quality control.
OnRenderEnd	Records information for quality control and synchronization.
OnRenderStart	Records information for quality control and synchronization.
OnStartStreaming	Resets all times that control streaming.
OnStopStreaming	Called at the end of streaming to fix times for the property page report.
OnWaitEnd	Called when a wait time ends. Performance logging only.
OnWaitStart	Updates times spent waiting and not waiting. Performance logging only.
RecordFrameLateness	Records how timely the rendering occurred and gathers statistics for the property page.
ResetStreamingTimes	Resets all times that control the streaming.
ScheduleSample	Sets up an advise link with the clock.
SendQuality	Sends a quality message to indicate what the supplier should do about quality.
ShouldDrawSampleNow	Determines if the video should be drawn when it is due, without setting a timer advise link with the clock.

Implemented IQualProp Methods

Name	Description
get_AvgFrameRate	Retrieves the average frame rate since streaming started in frames per 100 seconds.
get_AvgSyncOffset	Retrieves the average of the time in milliseconds between when each frame was due and when it was actually rendered. This applies to all frames since streaming started.
get_DevSyncOffset	Retrieves the standard deviation of the time in milliseconds between when each frame was due and when it was actually rendered for all frames since streaming started.
get_FramesDrawn	Retrieves the number of frames drawn since streaming started.
get_FramesDroppedInRenderer	Retrieves the number of frames dropped by the renderer. Frames can also be dropped upstream.
get_Jitter	Retrieves the standard deviation of the time in milliseconds between each frame and the next. This applies to all frames since streaming started.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingQueryInterface	Provides access to other interfaces, particularly the property page.

Implemented IQualityControl Methods

Name	Description
Notify	Notifies the recipient that a quality change is requested.
SetSink	Sets the IQualityControl object that will receive quality messages.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::CBaseVideoRenderer

CBaseVideoRenderer Class

Constructs a CBaseVideoRenderer object.

CBaseVideoRenderer(
 REFCLSID *RenderClass***,**
 TCHAR **pName***,**
 LPUNKNOWN *pUnk***,**
 HRESULT **phr*
 );

Parameters

RenderClass
    Class identifier for this renderer.
pName
    Description used for debugging purposes.
pUnk
    Pointer to the aggregated owner object.
phr
    Pointer to an HRESULT value.

Return Values

No return value.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::get_AvgFrameRate

CBaseVideoRenderer Class

Calculates and retrieves the average frame rate achieved.

HRESULT get_AvgFrameRate(
  int **piAvgFrameRate*
  );

Parameters

piAvgFrameRate
Number of frames per second since streaming began.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_AvgFrameRate method.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::get_AvgSyncOffset

CBaseVideoRenderer Class

Retrieves the average of the time in milliseconds between when each frame was due and when it was actually rendered for all frames since streaming started.

HRESULT get_AvgSyncOffset(
  int **piAvg*
  );

Parameters

piAvg
Pointer to the average of the time as previously described.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_AvgSyncOffset method.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::get_DevSyncOffset

CBaseVideoRenderer Class

Retrieves the standard deviation of the time in milliseconds between when each frame was due and when it was actually rendered, for all frames since streaming started.

HRESULT get_DevSyncOffset(
  int **piDev*
  );

Parameters

piDev
  Pointer to the standard deviation of the time as previously described.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_DevSyncOffset method.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::get_FramesDrawn

CBaseVideoRenderer Class

Retrieves the m_cFramesDrawn member variable, giving the number of frames drawn since streaming started.

HRESULT get_FramesDrawn(
  int **pcFramesDrawn*
  );

Parameters

pcFramesDrawn
Number of frames drawn.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_FramesDrawn method.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::get_FramesDroppedInRen

CBaseVideoRenderer Class

Retrieves the number of frames dropped by the renderer.

HRESULT get_FramesDroppedInRenderer(
  int **pcFramesDropped*
  );

Parameters

pcFramesDropped
Number of frames dropped.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_FramesDroppedInRenderer method. This is how the property page retrieves the data from the scheduler. Note that frames can also be dropped upstream without the renderer even seeing them.

CBaseVideoRenderer::get_Jitter

CBaseVideoRenderer Class

Retrieves the standard deviation of time in milliseconds between each frame and the next for all frames since streaming started.

HRESULT get_Jitter(
  int **piJitter*
  );

Parameters

piJitter
    Standard deviation of the interframe time in milliseconds.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualProp::get_Jitter method.

CBaseVideoRenderer::GetStdDev

CBaseVideoRenderer Class

Estimates the standard deviation in milliseconds between when each frame is due and when it is actually rendered, for per-frame statistics.

HRESULT GetStdDev(
  int *nSamples*,
  int **piResult*,
  LONGLONG *llSumSq*,
  LONGLONG *iTot*

);

Parameters

nSamples
: Integer value that contains the number of video samples received by the video renderer.

piResult
: Pointer to an integer value that will contain the standard deviation.

llSumSq
: Value that represents the standard deviation, in milliseconds, of all rendered video samples. The lower the value, the more consistent the rendering.

iTot
: Value that represents the mean value, in milliseconds, between the stamped time and rendered time for all rendered video samples.

Return Values

Returns NOERROR.

CBaseVideoRenderer::JoinFilterGraph

CBaseVideoRenderer Class

Sends EC_WINDOW_DESTROYED event notification when a filter is removed from the filter graph.

HRESULT JoinFilterGraph(
  IBaseFilterGraph * *pGraph***,**
  LPCWSTR *pName*
  );

Parameters

pGraph
: Pointer to the filter graph to join.

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

Return Values

No return value.

Remarks

This member function overrides the CBaseFilter::JoinFilterGraph member function. If this function determines that the filter is being notified that it is leaving the filter graph (*pGraph* is null, but m_pGraph is not), it sends an EC_WINDOW_DESTROYED event notification so that the resource manager does not hold on to the renderer as a focus object.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::NonDelegatingQueryInterf

CBaseVideoRenderer 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

Override this member function to publish the interface.

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method. It exposes the IQualProp interface and then calls CBaseRenderer::NonDelegatingQueryInterface to expose interfaces implemented in the base classes.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::Notify

CBaseVideoRenderer Class

Receives a notification that a quality change is requested.

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

Parameters

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

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualityControl::Notify method on the video renderer. This is called, typically by the filter graph manager, when the quality must be cut back. This might occur when the quality of audio playback has been increased to the point that the video playback quality must be decreased.

Notify sets the m_trThrottle data member to a delay value to be inserted between frames by ThrottleWait.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::OnDirectRender

CBaseVideoRenderer Class

Collects timing information that controls synchronization and quality control.

virtual void OnDirectRender(

```
  IMediaSample *pMediaSample
  );
```

Parameters

pMediaSample
Media sample.

Return Values

Returns an HRESULT value.

Remarks

Call this member function instead of OnRenderStart and OnRenderEnd. This is used by the Microsoft® DirectDraw® video renderer.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::OnRenderEnd

CBaseVideoRenderer Class

Performs smoothing for cases where the rendering time varies due to interruptions.

```
void OnRenderEnd(
  IMediaSample *pMediaSample
  );
```

Parameters

pMediaSample
Media sample.

Return Values

No return value.

Remarks

This member function should be called just after drawing an image.

This member function overrides CBaseRenderer::OnRenderEnd.

CBaseVideoRenderer::OnRenderStart

CBaseVideoRenderer Class

Sets information for rendering.

void OnRenderStart(
 IMediaSample **pMediaSample*
 );

Parameters

pMediaSample
Media sample.

Return Values

No return value.

Remarks

This member function retrieves the current clock time from the system and stores it in a member variable to be used when the drawing is complete. The function also performs performance logging. This member function should be called just before drawing starts.

This member function overrides CBaseRenderer::OnRenderStart.

CBaseVideoRenderer::OnStartStreaming

CBaseVideoRenderer Class

Resets all times that control streaming.

HRESULT OnStartStreaming(void);

Return Values

Returns an HRESULT value.

Remarks

This member function overrides CBaseRenderer::OnStartStreaming.

CBaseVideoRenderer::OnStopStreaming

CBaseVideoRenderer Class

Called at the end of streaming to fix times for the property page report.

HRESULT OnStopStreaming(void);

Return Values

Returns an HRESULT value.

Remarks

This member function is called twice, once when pausing and again when the stream is actually stopped.

This member function overrides CBaseRenderer::OnStopStreaming.

CBaseVideoRenderer::OnWaitEnd

CBaseVideoRenderer Class

Called when a wait time ends.

void OnWaitEnd(void);

Return Values

No return value.

Remarks

This member function does only performance logging. It is called when the thread is awoken from waiting in the window, or when the next sample is due to be rendered. It could eventually be used to gather information that controls synchronization.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::OnWaitStart

CBaseVideoRenderer Class

Updates times spent waiting and not waiting.

void OnWaitStart(void);

Return Values

No return value.

Remarks

This member function is called when starting to wait for a rendering event. It is used only for performance measurements.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::PreparePerformanceData

CBaseVideoRenderer Class

Sets the m_trLate and m_trFrame values of the current frame.

void PreparePerformanceData(
  int *trLate*,
  int *trFrame*

);

Parameters

trLate
How late the sample was beyond its due time, in reference time units.
trFrame
Interframe time, in reference time units.

Return Values

No return value.

Remarks

This member function sets m_trLate to the value of *trLate* and m_trFrame to the value of *trFrame*.

When the CBaseVideoRenderer::RecordFrameLateness member function is called from either CBaseVideoRenderer::OnRenderStart or CBaseVideoRenderer::OnDirectRender, it passes the values of m_trLate and m_trFrame for it to update the statistics. **PreparePerformanceData** is called from CBaseVideoRenderer::OnWaitEnd to set these data member values.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::RecordFrameLateness

CBaseVideoRenderer Class

Records how timely the rendering occurred and gathers statistics for the property page.

virtual void RecordFrameLateness(
int *trLate*,
int *trFrame*
);

Parameters

trLate
How late the sample was beyond its due time, in reference time units.
trFrame
Interframe time, in reference time units.

Return Values

No return value.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::ResetStreamingTimes

CBaseVideoRenderer Class

Resets all times that control the streaming.

virtual HRESULT ResetStreamingTimes(void);

Return Values

Returns an HRESULT value.

Remarks

The times are set so that frames will not be initially dropped and so that the first frame will be drawn.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::ScheduleSample

CBaseVideoRenderer Class

Overrides the base class that does the main work to keep a count of samples drawn and dropped (which are used by the IQualProp implementation).

BOOL ScheduleSample(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
  Media sample.

Return Values

Returns TRUE if the sample is scheduled; otherwise, returns FALSE.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::SendQuality

CBaseVideoRenderer Class

Sends a quality message to indicate what the supplier should do about quality.

virtual HRESULT SendQuality(
  REFERENCE_TIME *trLate*,
  REFERENCE_TIME *trRealStream*
  );

Parameters

trLate
  Amount of time by which the frame is late.
trRealStream
  Current stream time.

Return Values

Returns an HRESULT value.

Remarks

This member function sends a quality control message upstream to control quality management. The nature of the quality message (that is, whether to reduce or increase the number of samples) is determined in the quality management implementation in this derived class (see CBaseVideoRenderer::ShouldDrawSampleNow).

Previous Home Topic Contents Index Next

CBaseVideoRenderer::SetSink

CBaseVideoRenderer Class

Sets the IQualityControl object that will receive quality messages.

HRESULT SetSink(
 IQualityControl **piqc*
 );

Parameters

piqc
 Pointer to the IQualityControl object to which the notifications should be sent.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IQualityControl::SetSink method on the video renderer.

CBaseVideoRenderer::ShouldDrawSampleNow

CBaseVideoRenderer Class

Determines if the video should be drawn without setting a timer advise link with the clock.

virtual HRESULT ShouldDrawSampleNow(
 IMediaSample **pMediaSample*,
 REFERENCE_TIME **ptrStart*,
 REFERENCE_TIME **ptrEnd*
 );

Parameters

pMediaSample
 IMediaSample interface for the sample.
ptrStart
 Time to begin rendering.
ptrEnd
 Time to stop rendering.

Return Values

Returns an HRESULT value. Returns S_OK to mean draw at once without waiting, S_FALSE to mean draw at time *ptrStart*, or error to mean do not draw the sample; that is, skip it to save time.

Remarks

This member function overrides CBaseRenderer::ShouldDrawSampleNow.

Previous Home Topic Contents Index Next

CBaseVideoRenderer::ThrottleWait

CBaseVideoRenderer Class

Inserts a wait period after each frame.

void ThrottleWait(void);

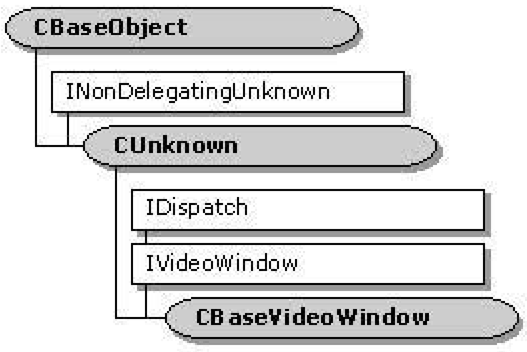
Return Values

No return value.

Remarks

This member function waits for a time period obtained from the m_trThrottle data member.

CBaseVideoWindow Class



The **CBaseVideoWindow** class handles the IDispatch component of the IVideoWindow interface and leaves the **IVideoWindow** properties and methods pure virtual.

Member Functions

Name	Description
CBaseVideoWindow	Constructs a CBaseVideoWindow object.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingQueryInterface	Returns a specified reference-counted interface.

Implemented IDispatch Methods

Name	Description
GetIDsOfNames	Maps a single member and an optional set of parameters to a corresponding set of integer dispatch identifiers, which can be used during subsequent calls to the IDispatch::Invoke method.
GetTypeInfo	Retrieves a type-information object, which can retrieve the type information for an interface.
GetTypeInfoCount	Retrieves the number of type-information interfaces provided by an object.
Invoke	Provides access to properties and methods exposed by an object.

CBaseVideoWindow::CBaseVideoWindow

CBaseVideoWindow Class

Constructs a CBaseVideoWindow object.

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

Parameters

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

pUnk
: Pointer to the owner of this object.

Return Values

No return value.

Remarks

Allocate the *pName* parameter in static memory. This name appears on the debugging terminal upon creation and deletion of an object.

Previous Home Topic Contents Index Next

CBaseVideoWindow::GetIDsOfNames

CBaseVideoWindow Class

Maps a single member function and an optional set of parameters to a corresponding set of integer dispatch identifiers, which can be used upon subsequent calls to the CBaseVideoWindow::Invoke member function.

HRESULT GetIDsOfNames(
  REFIID *riid*,
  OLECHAR ** *rgszNames*,
  UINT *cNames*,
  LCID *lcid*,
  DISPID * *rgdispid*
  );

Parameters

riid
    Reference identifier. Reserved for future use. Must be NULL.
rgszNames
    Passed-in array of names to be mapped.
cNames
    Count of the names to be mapped.
lcid
    Locale context in which to interpret the names.
rgdispid
    Caller-allocated array, each element of which contains an ID corresponding to one of the names passed in the *rgszNames* array. The first element represents the member name; the subsequent elements represent each of the member's parameters.

Return Values

Returns one of the following values.

Value	**Meaning**
DISP_E_UNKNOWN_CLSID	The CLSID was not recognized.
DISP_E_UNKNOWNNAME	One or more of the names were not known. The returned DISPIDs contain DISPID_UNKNOWN for each entry that corresponds to an unknown name.
E_OUTOFMEMORY	Out of memory.
S_OK	Success.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseVideoWindow::GetTypeInfo

CBaseVideoWindow Class

Retrieves a type-information object, which can retrieve the type information for an interface.

HRESULT GetTypeInfo(
  UINT *itinfo*,
  LCID *lcid*,
  ITypeInfo ** *pptinfo*
  );

Parameters

itinfo
: Type information to return. Pass zero to retrieve type information for the IDispatch implementation.

lcid
: Locale ID for the type information. An object might be able to return different type information for different languages. This is important for classes that support localized member names. For classes that do not support localized member names, this parameter can be ignored.

pptinfo
: Pointer to the type-information object requested.

Return Values

Returns an E_POINTER if *pptinfo* is invalid. Returns TYPE_E_ELEMENTNOTFOUND if *itinfo* is not zero. Returns S_OK if is successful. Otherwise, returns an HRESULT from one of the calls to retrieve the type. The **HRESULT** indicates the error and 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.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseVideoWindow::GetTypeInfoCount

CBaseVideoWindow Class

Retrieves the number of type-information interfaces provided by an object.

HRESULT GetTypeInfoCount(
  UINT * *pctinfo*
  );

Parameters

pctinfo
: Pointer to the location that receives the number of type-information interfaces that the object provides. If the object provides type information, this number is 1; otherwise, the number is 0.

Return Values

Returns E_POINTER if *pctinfo* is invalid; otherwise, returns S_OK.

CBaseVideoWindow::Invoke

CBaseVideoWindow Class

Provides access to properties and methods exposed by an object.

HRESULT Invoke(
  DISPID *dispidMember*,
  REFIID *riid*,
  LCID *lcid*,
  WORD *wFlags*,
  DISPPARAMS * *pdispparams*,
  VARIANT * *pvarResult*,
  EXCEPINFO * *pexcepinfo*,
  UINT * *puArgErr*
  );

Parameters

dispidMember
: Identifier of the member. Use CBaseVideoWindow::GetIDsOfNames or the object's documentation to obtain the dispatch identifier.

riid
: Reserved for future use. Must be IID_NULL.

lcid
: Locale context in which to interpret arguments.

wFlags
: Flags describing the context of the **CBaseVideoWindow::Invoke** call.

pdispparams
: Pointer to a structure containing an array of arguments, an array of argument dispatch IDs for named arguments, and counts for number of elements in the arrays.

pvarResult
: Pointer to where the result is to be stored, or NULL if the caller expects no result.

pexcepinfo
: Pointer to a structure containing exception information.

puArgErr
: Index of the first argument, within the *rgvarg* array, that has an error.

Return Values

Returns DISP_E_UNKNOWNINTERFACE if *riid* is not IID_NULL. Returns one of the error codes from CBaseVideoWindow::GetTypeInfo if the call fails. Otherwise, returns the HRESULT from the call to IDispatch::Invoke.

CBaseVideoWindow::NonDelegatingQueryInterfa

CBaseVideoWindow Class

Returns a specified reference-counted interface.

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

Returns pointers to the IVideoWindow and IUnknown interfaces by default. Override this method to publish any additional interfaces implemented by the derived class.

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method.

CBaseWindow Class

CBaseWindow

The **CBaseWindow** class creates a window and a worker thread. The worker thread pulls messages from the window's input queue and dispatches them as appropriate. The window and its thread are created by the CBaseWindow::PrepareWindow member function and destroyed by the CBaseWindow::DoneWithWindow member function. The window should also be initialized by the CBaseWindow::InitialiseWindow member function and uninitialized by the CBaseWindow::UninitialiseWindow member function. After preparing and initializing a window, size it by using the CBaseWindow::ActivateWindow member function; hide the window using the CBaseWindow::InactivateWindow member function.

Protected Data Members

Name	Description
m_bActivated	Flag to indicate window activation status.
m_bBackground	Flag to indicate if palettes are to be realized in the background.
m_bDoGetDC	Flag to indicate if the window should get a DC.
m_ClassStyles	Class styles for the window.
m_hdc	Device context (DC) for the window.
m_Height	Client window height.
m_hInstance	Global module instance handle.
m_hPalette	Handle to a palette belonging to this object.
m_hThread	Worker thread for the window.
m_hwnd	Handle for this object's window.
m_MemoryDC	Memory DC used for fast bit-block transfer operations.
m_pClassName	Static string holding the class name.
m_RealizePalette	Message sent to indicate the window palette has changed.
m_ShowStageMessage	Message sent by IVideoWindow::SetWindowForeground that moves the current window to the foreground and optionally gives it focus.
m_ShowStageTop	Message sent to set the window to WS_EX_TOPMOST style.
m_SyncWorker	CAMEvent data member used to provide interthread synchronization.
m_SyncWorkerCreate	CAMMsgEvent data member used to signal the constructor for the window class when to create the window.
m_ThreadSignal	Data member used by the thread to signal errors.
m_Width	Client window width.
m_WindowLock	Data member used to serialize window object access.
m_WindowStyles	Data member used to serialize the initial window styles.
m_WindowStylesEx	Data member used to serialize the initial extended window styles.

Member Functions

Name	Description
CBaseWindow	Constructs a CBaseWindow object.
DoSetWindowForeground	Brings the window to the foreground.
DoShowWindow	Sets the show state of the specified window.
GetMemoryHDC	Retrieves the default offscreen memory device context (DC).
GetWindowHDC	Retrieves the default main window DC.
GetWindowHeight	Retrieves the current window height.
GetWindowHWND	Retrieves the window handle for the window.
GetWindowWidth	Retrieves the current window width.
PerformanceAlignWindow	Aligns the window to a DWORD boundary for maximum performance.
PaintWindow	Invalidates the window client area.

Overridable Member Functions

Name	Description
ActivateWindow	Sizes the window according to the requirements of the derived class.
DoneWithWindow	Closes, deletes, and frees the window resources.
DoRealisePalette	Maps palette entries from this window's palette to the system palette. The window's palette is set with CBaseWindow::SetPalette.
PossiblyEatMessage	Forwards keyboard and mouse messages to a specified window.
GetClassWindowStyles	Returns class and window information.
GetDefaultRect	Returns the default size for the window.
InactivateWindow	Hides the window.
InitialiseWindow	Creates the default device contexts.
OnClose	Handles the WM_CLOSE message for the base class.
OnPaletteChange	Handles WM_PALETTEISCHANGING and WM_PALETTECHANGED messages.
OnSize	Handles WM_SIZE messages for the base class.
OnReceiveMessage	Indicates a base class implementation of a window procedure.
PrepareWindow	Initializes the window along with a worker thread.
SetPalette	Changes the palette that the window should realize.
UninitialiseWindow	Destroys the device contexts created for the window.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseWindow::ActivateWindow

CBaseWindow Class

Sizes the window according to the requirements of the derived class.

virtual HRESULT ActivateWindow( );

Return Values

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

Value	**Meaning**
E_FAIL	Failure.
E_NOTIMPL	Method is not supported.
NOERROR	No error.

Remarks

This member function calls CBaseWindow::GetDefaultRect, which a derived class should override to return the size of the images that will be displayed. **ActivateWindow** then sizes the window so that the client area matches this size.

CBaseWindow::CBaseWindow

CBaseWindow Class

Constructs a CBaseWindow object.

CBaseWindow(
  BOOL *bDoGetDC* **= TRUE**
  );

Parameters

bDoGetDC
    Specifies if the window should get a device context.

Return Values

No return value.

Remarks

The window and its worker thread are created by CBaseWindow::PrepareWindow and destroyed by CBaseWindow::DoneWithWindow.

Previous Home Topic Contents Index Next

CBaseWindow::DoneWithWindow

CBaseWindow Class

Destroys the window and its worker thread.

virtual HRESULT DoneWithWindow( );

Return Values

Returns an HRESULT value. Current implementation returns NOERROR.

Remarks

The base window class creates a window and a worker thread. The worker thread is responsible for pulling messages from the window's input queue and dispatching them as appropriate. The window and its thread are created by CBaseWindow::PrepareWindow and destroyed by **CBaseWindow::DoneWithWindow**. The window should also be initialized using CBaseWindow::InitialiseWindow and uninitialized using CBaseWindow::UninitialiseWindow. Having prepared a window and initialized it, the window can be sized using CBaseWindow::ActivateWindow and subsequently hidden using CBaseWindow::InactivateWindow.

Previous Home Topic Contents Index Next

CBaseWindow::DoRealisePalette

CBaseWindow Class

Maps palette entries from this window's palette to the system palette. The window's palette is set with CBaseWindow::SetPalette.

virtual HRESULT DoRealisePalette(
  BOOL *bForceBackground*
  );

Parameters

bForceBackground
Value that specifies whether the palette is forced to the background.

Return Values

Returns S_OK if successful or S_FALSE if the GdiFlush function could not flush the calling thread's current batch.

Remarks

The window class is given a palette handle to use with the CBaseWindow::SetPalette member function. After a palette has been installed, it can be realized by calling this member function. The class will also call this member function when it gets WM_QUERYNEWPALETTE and WM_PALETTECHANGED messages from the Microsoft® Windows® operating system.

Call this function with TRUE in response to WM_SETPALETTE and FALSE in response to WM_QUERYNEWPALETTE.

Previous Home Topic Contents Index Next

CBaseWindow::DoSetWindowForeground

CBaseWindow Class

Sets the video window to the foreground and optionally gives it focus.

void DoSetWindowForeground(
BOOL *bFocus*
);

Parameters

bFocus
Value that specifies whether the video window will have focus. A value of TRUE gives it focus and FALSE does not.

Return Values

No return value.

Remarks

DirectShow provides this method to make it easy for applications to move video windows to the foreground; usually, it is programatically complex for a thread associated with one window to affect a window associated with a different thread. This method passes the WM_SHOWWINDOW message to the video window's renderer, so the application's window procedure must handle this message and bring the appropriate window to the foreground and give it focus, if specified.

CBaseWindow::DoShowWindow

CBaseWindow Class

Sets the show state of the specified window.

HRESULT DoShowWindow(
  LONG *ShowCmd*
  );

Parameters

ShowCmd
  Specifies how the window is to be shown.

Return Values

Returns an HRESULT value. Current implementation returns NOERROR.

Remarks

This member function simply calls the Microsoft Win32® ShowWindow function.

CBaseWindow::GetClassWindowStyles

CBaseWindow Class

Returns class and window information.

virtual LPTSTR GetClassWindowStyles(
  DWORD **pClassStyles*,
  DWORD **pWindowStyles*,
  DWORD **pWindowStylesEx*
) PURE;

Parameters

pClassStyles
    Class styles.
pWindowStyles
    Window styles.
pWindowStylesEx
    Extended window styles.

Return Values

Returns a class name that is a static text string.

Remarks

A derived class must override this pure virtual member function to provide the default class and window styles for the window. The information the derived class returns is used in CBaseWindow::PrepareWindow when the window is first created. The class and window styles take the same parameters as their counterparts in the Microsoft Win32 CreateWindowEx function. The string that is returned should be allocated as a static string and should still be valid after the member function returns.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseWindow::GetDefaultRect

CBaseWindow Class

Retrieves the default size for the window client area.

virtual RECT GetDefaultRect( );

Return Values

Returns the default rectangle.

Remarks

When the window is activated, it calls this member function to determine how large it should make the window's client area. A video renderer will typically return the size of the native video image.

Previous Home Topic Contents Index Next

CBaseWindow::GetMemoryHDC

CBaseWindow Class

Retrieves the default memory device context (DC).

virtual HDC GetMemoryHDC( );

Return Values

Returns the default memory DC.

Remarks

The base window class creates a window with a worker thread when it is prepared (in CBaseWindow::PrepareWindow). It also creates two DCs that can be used for drawing. The first is a normal window handle to a device context (HDC); the second is an offscreen HDC that can be used as a source HDC in bit-block transfer functions.

Previous Home Topic Contents Index Next

CBaseWindow::GetWindowHDC

CBaseWindow Class

Retrieves the default window device context (DC).

HDC GetWindowHDC( );

Return Values

Returns the default window DC.

Remarks

The base window class creates a window with a worker thread when it is prepared (in CBaseWindow::PrepareWindow). It also creates two DCs that can be used for drawing. The first is a normal window handle to a device context (HDC); the second is an offscreen HDC that can be used as a source HDC in bit-block transfer functions.

Previous Home Topic Contents Index Next

CBaseWindow::GetWindowHeight

CBaseWindow Class

Retrieves the current window height.

LONG GetWindowHeight( );

Return Values

Returns the window height in pixels.

Remarks

This member function is updated when the base class receives WM_SIZE messages.

Previous Home Topic Contents Index Next

CBaseWindow::GetWindowHWND

CBaseWindow Class

Retrieves the window handle associated with this object.

HWND GetWindowHWND( );

Return Values

Returns a window handle.

Remarks

If called before issuing a CBaseWindow::PrepareWindow call, this member function returns NULL.

Previous Home Topic Contents Index Next

CBaseWindow::GetWindowWidth

CBaseWindow Class

Retrieves the current window width.

LONG GetWindowWidth( );

Return Values

Returns the window width in pixels.

Remarks

This member function is updated when the base class receives WM_SIZE messages.

Previous Home Topic Contents Index Next

CBaseWindow::InactivateWindow

CBaseWindow Class

Effectively hides the window (if it was visible).

virtual HRESULT InactivateWindow( );

Return Values

Returns NOERROR if successful; S_FALSE if the window is not currently active.

Remarks

The base window class creates a window and a worker thread. The worker thread is responsible for pulling messages from the window's input queue and dispatching them as appropriate. The window and its thread are created by CBaseWindow::PrepareWindow and destroyed in CBaseWindow::DoneWithWindow. The window should be initialized through CBaseWindow::InitialiseWindow and uninitialized through CBaseWindow::UninitialiseWindow. Having prepared a window and initialized it, the window can be sized using CBaseWindow::ActivateWindow and subsequently hidden using **CBaseWindow::InactivateWindow**.

Previous Home Topic Contents Index Next

CBaseWindow::InitialiseWindow

CBaseWindow Class

Creates default device contexts for the window.

virtual InitialiseWindow(
  HWND *hwnd*
  );

Parameters

hwnd
  Window handle.

Return Values

Returns an HRESULT value. Current implementation returns NOERROR.

Remarks

The base window class creates a window and a worker thread. The worker thread is responsible for pulling messages from the window's input queue and dispatching them as appropriate. The window and its thread are created by CBaseWindow::PrepareWindow and destroyed in CBaseWindow::DoneWithWindow. The window should be initialized through **CBaseWindow::InitialiseWindow** and uninitialized through CBaseWindow::UninitialiseWindow. Having prepared a window and initialized it, the window can be sized using CBaseWindow::ActivateWindow and subsequently hidden using CBaseWindow::InactivateWindow.

The base class creates two device contexts that can be used for drawing. The first is a standard handle to a device context (HDC) for the window; the second is an offscreen HDC. The offscreen HDC often is useful for selecting bitmaps before calling the Microsoft Win32 BitBlt or StretchBlt function to copy the bitmap to the main window. This member function also sets the default stretch mode to be COLORONCOLOR. The member function is virtual so that derived classes can change this default if desired.

Previous Home Topic Contents Index Next

CBaseWindow::OnClose

CBaseWindow Class

Handles the WM_CLOSE message.

virtual BOOL OnClose( );

Return Values

No return value.

Remarks

The default behavior for this member function is to simply hide the window. A derived class should not destroy the window when it receives a WM_CLOSE message but should send an EC_USERABORT notification to the filter graph manager. This will have the playback stopped, and in some cases will also have the filters disconnected and released. It is only when the filter that owns the window is finally released (that is, destroyed) that the derived class should actually destroy the window (using CBaseWindow::DoneWithWindow).

Previous Home Topic Contents Index Next

CBaseWindow::OnPaletteChange

CBaseWindow Class

Handles WM_PALETTEISCHANGING and WM_PALETTECHANGED messages.

virtual LRESULT OnPaletteChange(
  HWND *hwnd*,
  UINT *Message*
  );

Parameters

hwnd
Handle of the window causing the message.
Message
Message details passed on from the window procedure.

Return Values

Returns one of the following values.

Value	Meaning
0	Message was not handled.
1	Message was processed.

Remarks

When the base class receives a WM_PALETTEISCHANGING message, it realizes its palette again. It must also do this when told, through WM_PALETTECHANGED, that the system palette has changed. In the latter case, however, the base class must be careful not to realize its palette if it was the window that caused the WM_PALETTECHANGED message (which is why the window that caused the message to be sent is passed into the member function).

This is a protected member function.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseWindow::OnReceiveMessage

CBaseWindow Class

Indicates a base class implementation of a window procedure.

virtual LRESULT OnReceiveMessage(
  HWND *hwnd*,
  INT *uMsg*,
  WPARAM *wParam*,
  LPARAM *lParam*
  );

Parameters

hwnd
Handle to the window.
uMsg

Message identifier.
wParam
Message's *wParam* parameter.
lParam
Message's *lParam* parameter.

Return Values

Returns an LRESULT value, based on the *uMsg* parameter. If *uMsg* is not one of the specified values, **OnReceiveMessage** passes the message to the Win32 DefWindowProc function and forwards the resulting return value to the caller.

Message	**Action**
m_RealizePalette	Returns 0
m_ShowStageMessage	Returns 1
m_ShowStageTop	Returns 1
WM_CLOSE	Returns 0
WM_PALETTECHANGED	Returns 0
WM_QUERYNEWPALETTE	Returns result of CBaseWindow::OnPaletteChange
WM_SIZE	Returns 0
WM_SYSCOLORCHANGE	Returns 1

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CBaseWindow::OnSize

CBaseWindow Class

Handles the WM_SIZE message.

virtual BOOL OnSize(
  LONG *Width*,
  LONG *Height*
  );

Parameters

Width
Window width.
Height
Window height.

Return Values

No return value.

Remarks

This member function stores the window width and height so that they can be returned from the CBaseWindow::GetWindowHeight and CBaseWindow::GetWindowWidth member functions.

Previous Home Topic Contents Index Next

CBaseWindow::PaintWindow

CBaseWindow Class

Invalidates the window client area.

void PaintWindow(
 BOOL *bErase*
 );

Parameters

bErase
 Determines if the background should be erased.

Return Values

No return value.

Previous Home Topic Contents Index Next

CBaseWindow::PerformanceAlignWindow

CBaseWindow Class

Aligns the window to a DWORD boundary for maximum performance.

HRESULT PerformanceAlignWindow( );

Return Values

Returns an HRESULT value.

Remarks

This member function can be called, if the video is not owned by another window, to align the left edge and the top of the window for best display performance.

Previous Home Topic Contents Index Next

CBaseWindow::PossiblyEatMessage

CBaseWindow Class

Forwards keyboard and mouse messages to a specified window.

virtual BOOL PossiblyEatMessage(
  UINT *uMsg*,
  WPARAM *wParam*,
  LPARAM *lParam*
  );

Parameters

uMsg
    Message that was forwarded.
wParam
    First message parameter.
lParam
    Second message parameter.

Return Values

Returns FALSE.

See Also

CBaseControlWindow::PossiblyEatMessage

Previous Home Topic Contents Index Next

CBaseWindow::PrepareWindow

CBaseWindow Class

Creates a window and a worker thread.

virtual HRESULT PrepareWindow( );

Return Values

Returns NOERROR if successful; E_FAIL if unsuccessful.

Remarks

The base window class creates a window and a worker thread. The worker thread is responsible for pulling messages from the window's input queue and dispatching them as appropriate. The window and its thread are created by **CBaseWindow::PrepareWindow** and destroyed in CBaseWindow::DoneWithWindow. The window should also be initialized and uninitialized through CBaseWindow::InitialiseWindow and CBaseWindow::UninitialiseWindow, respectively. Having prepared a window and initialized it, the window can be sized using CBaseWindow::ActivateWindow and subsequently hidden using CBaseWindow::InactivateWindow.

CBaseWindow::SetPalette

CBaseWindow Class

Sets a palette for the window to use.

virtual HRESULT SetPalette(
  HPALETTE *hPalette*
  );

Parameters

hPalette

Handle to the new palette.

Return Values

Returns an HRESULT value.

Remarks

This member function allows a filter to install a palette in the window object. The palette handle passed in should be non-NULL. The palette is realized when it is installed. The window object does not delete any previous palette that it was using; the client using the window object should ensure it deletes the palette it creates at the appropriate time.

Previous Home Topic Contents Index Next

CBaseWindow::UninitialiseWindow

CBaseWindow Class

Flushes GDI and deletes the default device contexts.

virtual UninitialiseWindow( );

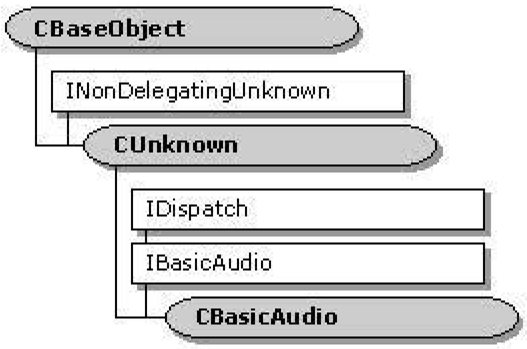
Return Values

Returns an HRESULT value. Current implementation returns NOERROR.

Remarks

The base window class creates a window and a worker thread. The worker thread is responsible for pulling messages from the window's input queue and dispatching them as appropriate. The window and its thread are created by CBaseWindow::PrepareWindow and destroyed in CBaseWindow::DoneWithWindow. The window should also be initialized and uninitialized through CBaseWindow::InitialiseWindow and **CBaseWindow::UninitialiseWindow**, respectively. Having prepared a window and initialized it, the window can be sized using CBaseWindow::ActivateWindow and subsequently hidden using CBaseWindow::InactivateWindow.

CBasicAudio Class



The **CBasicAudio** class handles the IDispatch interface component of the IBasicAudio interface and leaves the properties and methods of **IBasicAudio** pure virtual to be implemented by a derived filter class.

The CBasicAudio::GetIDsOfNames, CBasicAudio::GetTypeInfo, CBasicAudio::GetTypeInfoCount, and CBasicAudio::Invoke member functions are standard implementations of the IDispatch interface using the CBaseDispatch class (and a type library) to parse the commands and pass them to the pure virtual IBasicAudio methods.

Microsoft® DirectShow™ uses units of 100th of a decibel for the volume scale. A value of 0 indicates maximum volume supported by the device. A value of –10,000 is the minimum volume (normally silence). Balance is expressed in the range –10,000 to 10,000, with 0 being neutral. A negative balance value means that the right channel is attenuated by this dB value (that is, it is quieter). Similarly, a positive balance value means that the right channel is louder than the left; that is, the left channel is attenuated by the corresponding negative decibel value.

Member Functions

Name	Description
CBasicAudio	Constructs a CBasicAudio object.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingQueryInterface	Returns a specified reference-counted interface.

Implemented IDispatch Methods

Name	Description
GetIDsOfNames	Maps a single member and an optional set of parameters to a corresponding set of integer dispatch identifiers, which can be used during subsequent calls to the CBasicAudio::Invoke member function.
GetTypeInfo	Retrieves a type-information object, which can retrieve the type information for an interface.
GetTypeInfoCount	Retrieves the number of type-information interfaces provided by an object.
Invoke	Provides access to properties and methods exposed by an object.

CBasicAudio::CBasicAudio

CBasicAudio Class

Constructs a CBasicAudio object.

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

Parameters

pName
  Name of the object used in the CBasicAudio constructor for debugging purposes.
pUnk
  Pointer to the owner of this object.

Return Values

No return value.

Remarks

Allocate the *pName* parameter in static memory. This name appears on the debugging terminal when the object is created and deleted.

CBasicAudio::GetIDsOfNames

CBasicAudio Class

Maps a single member function and an optional set of parameters to a corresponding set of integer dispatch identifiers, which can be used upon subsequent calls to the CBasicAudio::Invoke member function.

HRESULT GetIDsOfNames(
  REFIID *riid***,**
  OLECHAR ** *rgszNames***,**
  UINT *cNames***,**
  LCID *lcid***,**
  DISPID * *rgdispid*
  );

Parameters

riid
: Reference identifier. Reserved for future use. Must be NULL.

rgszNames
: Passed-in array of names to be mapped.

cNames
: Count of the names to be mapped.

lcid
: Locale context in which to interpret the names.

rgdispid
: Caller-allocated array, each element of which contains an ID corresponding to one of the names passed in the *rgszNames* array. The first element represents the member name; the subsequent elements represent each of the member's parameters.

Return Values

Returns one of the following values.

Value	**Meaning**
DISP_E_UNKNOWN_CLSID	The CLSID was not recognized.
DISP_E_UNKNOWNNAME	One or more of the names were not known. The returned DISPIDs contain DISPID_UNKNOWN for each entry that corresponds to an unknown name.
E_OUTOFMEMORY	Out of memory.
S_OK	Success.

Previous Home Topic Contents Index Next

CBasicAudio::GetTypeInfo

CBasicAudio Class

Retrieves a type-information object, which can retrieve the type information for an interface.

HRESULT GetTypeInfo(
  UINT *itinfo*,
  LCID *lcid*,
  ITypeInfo ** *pptinfo*
  );

Parameters

itinfo
: Type information to return. Pass zero to retrieve type information for the IDispatch implementation.

lcid
: Locale ID for the type information. An object might be able to return different type information for different languages. This is important for classes that support localized member names. For classes that do not support localized member names, this parameter can be ignored.

pptinfo
: Pointer to the type-information object requested.

Return Values

Returns an E_POINTER if *pptinfo* is invalid. Returns TYPE_E_ELEMENTNOTFOUND if *itinfo* is not zero. Returns S_OK if is successful. Otherwise, returns an HRESULT from one of the calls to retrieve the type. The **HRESULT** indicates the error and 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.

Previous Home Topic Contents Index Next

CBasicAudio::GetTypeInfoCount

CBasicAudio Class

Retrieves the number of type-information interfaces provided by an object.

HRESULT GetTypeInfoCount(
  UINT * *pctinfo*
  );

Parameters

pctinfo
: Pointer to the location that receives the number of type-information interfaces that the object provides. If the object provides type information, this number is 1; otherwise, the number is 0.

Return Values

Returns E_POINTER if *pctinfo* is invalid; otherwise, returns S_OK.

Previous Home Topic Contents Index Next

CBasicAudio::Invoke

CBasicAudio Class

Provides access to properties and methods exposed by an object.

HRESULT Invoke(
  DISPID *dispidMember*,
  REFIID *riid*,
  LCID *lcid*,
  WORD *wFlags*,
  DISPPARAMS * *pdispparams*,
  VARIANT * *pvarResult*,
  EXCEPINFO * *pexcepinfo*,
  UINT * *puArgErr*
  );

Parameters

dispidMember

Identifier of the member. Use CBasicAudio::GetIDsOfNames or the object's documentation to obtain the dispatch identifier.

riid
: Reserved for future use. Must be IID_NULL.

lcid
: Locale context in which to interpret arguments.

wFlags
: Flags describing the context of the **CBasicAudio::Invoke** call.

pdispparams
: Pointer to a structure containing an array of arguments, an array of argument dispatch IDs for named arguments, and counts for number of elements in the arrays.

pvarResult
: Pointer to where the result is to be stored, or NULL if the caller expects no result.

pexcepinfo
: Pointer to a structure containing exception information.

puArgErr
: Index of the first argument, within the *rgvarg* array, that has an error.

Return Values

Returns DISP_E_UNKNOWNINTERFACE if *riid* is not IID_NULL. Returns one of the error codes from CBasicAudio::GetTypeInfo if the call fails. Otherwise, returns the HRESULT from the call to IDispatch::Invoke.

CBasicAudio::NonDelegatingQueryInterface

CBasicAudio Class

Returns a specified reference-counted interface.

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

Returns pointers to the IBasicAudio and IUnknown interfaces by default. Override this member function to publish any additional interfaces implemented by the derived class.

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method.

CCmdQueue Class

The **CCmdQueue** class is a base class that provides a queue of CDeferredCommand objects and member functions to add, remove, check status, and invoke the queued commands. A **CCmdQueue** object is a part of an object that implements IQueueCommand methods. The filter graph manager implements **IQueueCommand** methods so that applications can queue commands to the filter graph. Filters that implement the **IQueueCommand** interface directly use this class. If you want to use **CDeferredCommand** objects, your queue must be derived from this class.

There are two modes of synchronization: coarse and accurate. In coarse mode, the application waits until a specified time arrives and then executes the command. In accurate mode, the application waits until processing begins on the sample that appears at the time, and then executes the command. The filter determines which one it will implement. The filter graph manager always implements coarse mode for commands that are queued at the filter graph manager.

If you want coarse synchronization, you probably want to wait until there is a command due, and then execute it. You can do this by calling CCmdQueue::GetDueCommand. If you have several things to wait for, get the event handle from CCmdQueue::GetDueHandle and then call **CCmdQueue::GetDueCommand** when this is signaled. Stream time will advance only between calls to the CCmdQueue::Run and CCmdQueue::EndRun member functions. There is no guarantee that if the handle is set, there will be a command ready. Each time the event is signaled, call the **CCmdQueue::GetDueCommand** member function (probably with a time-out of zero); this may return E_ABORT.

If you want accurate synchronization, call the CCmdQueue::GetCommandDueFor member function and pass the samples you are about to process as a parameter. This returns the following:

- A stream-time command due at or before that stream time.
- A presentation-time command due at or before the presentation of the stream time. Do this only between the CCmdQueue::Run and CCmdQueue::EndRun member functions, because outside of this, the mapping from stream time to presentation time is not known.
- Any presentation-time command due now.

If you want accurate synchronization for samples that might be processed during paused mode, you must use stream-time commands.

In all cases, commands remain queued until called or canceled. The setting and resetting of the event handle is managed entirely by this queue object.

Protected Data Members

Name	Description
m_bRunning	Flag for running state; set TRUE when running.
m_dwAdvise	Advise identifier from the reference clock (zero if no outstanding advise).
m_evDue	Sets the time when any commands are due.
m_listPresentation	Stores commands queued in presentation time.
m_listStream	Stores commands queued in stream time.
m_Lock	Protects access to lists.
m_pClock	Current reference clock.
m_StreamTimeOffset	Contains the stream time offset when m_bRunning is true.
m_tCurrentAdvise	Advise time is for this presentation time.

Member Functions

Name	Description
CCmdQueue	Constructs a CCmdQueue object.
CheckTime	Determines if a given time is due.
GetDueHandle	Returns the event handle that will be signaled.

Overridable Member Functions

Name	Description
EndRun	Switches to stopped or paused mode.
GetCommandDueFor	Returns a pointer to a command that will be due for a given time.
GetDueCommand	Returns a pointer to the next command that is due.
Insert	Adds the CDeferredCommand object to the queue.
New	Initializes a command to be run and returns a new CDeferredCommand object.
Remove	Removes the CDeferredCommand object from the queue.
Run	Switches to running mode.
SetSyncSource	Sets the clock used for timing.
SetTimeAdvise	Creates an advise for the earliest time required.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CCmdQueue::CCmdQueue

CCmdQueue Class

Constructs a CCmdQueue object.

CCmdQueue( );

Return Values

No return value.

Previous Home Topic Contents Index Next

CCmdQueue::CheckTime

CCmdQueue Class

Determines if a specified time is due.

BOOL CheckTime(
  CRefTime *time*,
  BOOL *bStream*
  );

Parameters

time
: Time to check.

bStream
: TRUE if the *time* parameter is a stream-time value; FALSE if *time* is a presentation-time value.

Return Values

Returns TRUE if the specified time has not yet passed.

Previous Home Topic Contents Index Next

CCmdQueue::EndRun

CCmdQueue Class

Switches to the stopped or paused mode.

virtual HRESULT EndRun( );

Return Values

Returns an HRESULT value that depends on the implementation. The **HRESULT** indicates the error and 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.
S_OK or NOERROR	Success.

Remarks

Time mapping between stream time and presentation time is not known after this member function has been called. Call the CCmdQueue::Run member function to carry out this mapping.

CCmdQueue::GetCommandDueFor

CCmdQueue Class

Returns a deferred command that is scheduled at a specified time.

virtual HRESULT GetCommandDueFor(
  REFERENCE_TIME *tStream*,
  CDeferredCommand ***ppCmd*
  );

Parameters

tStream
    Time for which the command is scheduled.
ppCmd
    Deferred command to be carried out at the time specified in the *tStream* parameter.

Return Values

Returns VFW_E_NOT_FOUND if no commands are due; otherwise, returns S_OK.

Remarks

This member function takes a stream time and returns the deferred command scheduled at that time. The actual stream-time offset is calculated when the command queue is run. Commands remain queued until run or canceled. This member function will not block.

Previous Home Topic Contents Index Next

CCmdQueue::GetDueCommand

CCmdQueue Class

Returns a pointer to the next command that is due.

virtual HRESULT GetDueCommand(
  CDeferredCommand ** *ppCmd*,
  long *msTimeout*
  );

Parameters

ppCmd
  Pointer to the deferred command.
msTimeout
  Amount of time to wait before carrying out the time-out.

Return Values

Returns E_ABORT if a time-out occurs. Returns S_OK if successful; otherwise, returns an error. Returns an object that has been incremented using IUnknown::AddRef.

Remarks

This member function blocks until a pending command is due. The member function blocks for the amount of time, in milliseconds, specified in the *msTimeout* parameter. Stream-time commands become due only between the CCmdQueue::Run and CCmdQueue::EndRun member functions. The command remains queued until run or canceled.

Previous Home Topic Contents Index Next

CCmdQueue::GetDueHandle

CCmdQueue Class

Returns the event handle to be signaled.

HANDLE GetDueHandle( );

Return Values

Returns the event handle.

Remarks

Return the event handle whenever there are deferred commands that are due for execution (when CCmdQueue::GetDueCommand will not block).

Previous Home Topic Contents Index Next

CCmdQueue::Insert

CCmdQueue Class

The CDeferredCommand object calls this member function to add itself to the queue.

virtual HRESULT Insert(
  CDeferredCommand* *pCmd*
  );

Parameters

pCmd
  Pointer to the CDeferredCommand object to add to the queue.

Return Values

Returns S_OK in the default implementation.

Previous Home Topic Contents Index Next

CCmdQueue::New

CCmdQueue Class

Initializes a command to be run and returns a new CDeferredCommand object.

virtual HRESULT New(
 CDeferredCommand ***ppCmd*,**
 LPUNKNOWN ***pUnk*,**
 REFTIME ***time*,**
 GUID* ***iid*,**
 long ***dispidMethod*,**
 short ***wFlags*,**
 long ***cArgs*,**
 VARIANT* ***pDispParams*,**
 VARIANT* ***pvarResult*,**
 short* ***puArgErr*,**
 BOOL ***bStream***
 );

Parameters

ppCmd
    CDeferredCommand object by which an application can cancel the command, set a new presentation time for it, or retrieve estimate information.
pUnk
    Pointer to the object that will run the command.
time
    Time at which to run the queued command or commands.
iid
    Globally unique identifier (GUID) of the interface to call.
dispidMethod
    Method on the interface to be called.
wFlags
    Flags describing the context of the call. This parameter supports the same flags as the OLE IDispatch::Invoke method.
cArgs
    Number of arguments passed.
pDispParams
    Pointer to the list of variant types associated with the dispatch parameters.
pvarResult
    Pointer to the list where results, if any, are to be returned.
puArgErr
    Index within the *pDispParams* parameter list where the last error occurred.
bStream
    TRUE if the *time* parameter is a stream-time value; FALSE if *time* is a presentation-time value.

Return Values

Returns S_OK if successful. Returns E_OUTOFMEMORY if *ppCmd* returns from creating the new

CDeferredCommand object with a value of NULL. Otherwise, returns an HRESULT that indicates an error from attempting to create a new **CDeferredCommand** object. If there is an error, no object has been queued.

Remarks

The new CDeferredCommand object will be initialized with the parameters and will be added to the queue during construction. This method is similar to the OLE IDispatch::Invoke method.

Values for the *wFlags* parameter include the following:

Value	**Description**
DISPATCH_METHOD	The member is being run as a method. If a property has the same name, both this and the DISPATCH_PROPERTYGET flag may be set.
DISPATCH_PROPERTYGET	The member is being retrieved as a property or data member.
DISPATCH_PROPERTYPUT	The member is being changed as a property or data member.
DISPATCH_PROPERTYPUTREF	The member is being changed via a reference assignment, rather than a value assignment. This value is valid only when the property accepts a reference to an object.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CCmdQueue::Remove

CCmdQueue Class

The CDeferredCommand object calls this member function to remove itself from the queue.

virtual HRESULT Remove(
 CDeferredCommand* *pCmd*
 );

Parameters

pCmd
 Pointer to the CDeferredCommand object to remove from the queue.

Return Values

Returns VFW_E_NOT_FOUND if the object is not found in the queue. Otherwise, returns S_OK.

CCmdQueue::Run

CCmdQueue Class

Switches to running mode so that commands that are deferred by the stream time can be run.

virtual HRESULT Run(
  REFERENCE_TIME *tStreamTimeOffset*
  );

Parameters

tStreamTimeOffset
  Offset time.

Return Values

Returns S_OK in the default implementation.

Remarks

During running mode, stream-time-to-presentation-time mapping is known.

CCmdQueue::SetSyncSource

CCmdQueue Class

Sets the clock used for timing.

virtual HRESULT SetSyncSource(
  IReferenceClock* *pIrc*
  );

Parameters

pIrc

Pointer to the IReferenceClock interface.

Return Values

Returns S_OK in the default implementation.

CCmdQueue::SetTimeAdvise

CCmdQueue Class

Sets up a timer event with the reference clock.

void SetTimeAdvise(void);

Return Values

No return value.

Remarks

This member function calls the IReferenceClock::AdviseTime method to set up a notification for the earliest time required in the queue. Presentation-time commands that are deferred are always checked. If the filter graph is in running mode, deferred commands using stream time are also checked.

CCritSec Class

CCritSec

The critical section object provides intraprocess synchronization. The current implementation uses the Microsoft® Win32® application programming interfaces (APIs) that use the CRITICAL_SECTION type.

The safest way to use **CCritSec** objects is to lock them with a CAutoLock object that guarantees to unlock the object when it goes out of scope and compiles to efficient inline code.

Member Functions

Name	Description
CCritSec	Constructs a CCritSec object.
Lock	Locks the critical section object.
Unlock	Unlocks the critical section object.

CCritSec::CCritSec

CCritSec Class

Constructs a CCritSec object.

CCritSec( );

Return Values

No return value.

Remarks

Calls the Microsoft® Win32® InitializeCriticalSection function to set the private critical section member variable. The destructor calls the Win32 DeleteCriticalSection function.

CCritSec::Lock

CCritSec Class

Locks the critical section object.

void Lock( );

Return Values

No return value.

Remarks

This member function locks the critical section object. You can make multiple lock calls on the same thread, but the CCritSec::Unlock member function must be called a corresponding number of times before the object is unlocked. If the object is locked by another thread, the **CCritSec::Lock** member function blocks until either the object is released or a "possible deadlock" exception occurs.

CCritSec::Unlock

CCritSec Class

Releases the lock on the object acquired by calling the CCritSec::Lock member function.

void Unlock( );

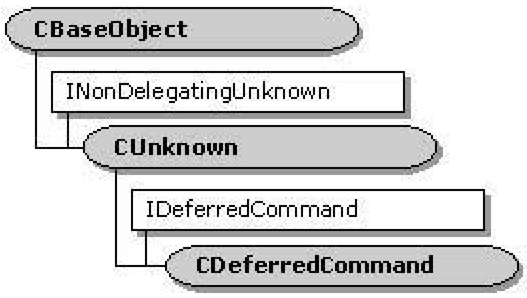
Return Values

No return value.

Remarks

You must call the **CCritSec::Unlock** member function once for each call to Lock.

CDeferredCommand Class



Deferred commands are queued by calls to methods on the IQueueCommand interface and are exposed by the filter graph manager and by some filters. A successful call to one of these methods returns an IDeferredCommand interface representing the queued command.

A **CDeferredCommand** object represents a single deferred command and exposes the IDeferredCommand interface as well as other methods that permit time checks and actual execution. A **CDeferredCommand** object contains a reference to the CCmdQueue object on which it is queued.

Reference counts control the lifetime of the **CDeferredCommand** class. When calling the CDeferredCommand::Invoke member function, the calling application gets an interface pointer that is reference-counted, and the CCmdQueue object also holds a reference count on the deferred command. Calling the IDeferredCommand::Cancel member function takes the deferred command off the command queue and thus reduces the reference count by one. Once taken off the queue, the command cannot be put back on the queue.

Protected Data Members

Name	**Description**
m_bStream	Flag for stream time or presentation time. to be passed to the invoked method.
m_Dispatch	Accesses the ITypeInfo interface.
m_dispidMethod	Method on the interface to run.
m_DispParams	CDispParams object containing the DISPPARAMS parameter list
m_hrResult	Stores the returned HRESULT value.
m_iid	Globally unique identifier (GUID) of the interface.
m_pQueue	Pointer to the CCmdQueue object that exposes the IQueueCommand interface.
m_pUnk	IUnknown pointer to the interface on which the command will be run.
m_pvarResult	Resulting information, if any, from the invoked method.
m_time	Time at which the command will be run.
m_wFlags	Flags specifying the context of the invocation.

Member Functions

Name	Description
CDeferredCommand	Constructs a CDeferredCommand object.
GetFlags	Returns the context flags associated with the deferred command.
GetIID	Returns the interface identifier (IID) of the interface on which the method will be run.
GetMethod	Returns the dispatch identifier of the method to be run.
GetParams	Returns the DISPPARAMS argument list to the method.
GetResult	Returns the resulting argument list, if one exists.
GetTime	Returns the time when the method will be run.
Invoke	Provides access to methods and properties exposed by an object.
IsStreamTime	Specifies whether the command is to be run at stream time or presentation time.

Implemented IDeferredCommand Methods

Name	Description
Cancel	Cancels a previously queued CDeferredCommand::Invoke request.
Confidence	Not currently implemented.
Postpone	Specifies a new presentation time for a previously queued command.
GetHResult	Returns the HRESULT value of the invoked method.

Implemented INonDelegatingUnknown Methods

Name	Description
NonDelegatingQueryInterface	Returns a specified reference-counted interface.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CDeferredCommand::Cancel

CDeferredCommand Class

Cancels a previously queued CDeferredCommand::Invoke request.

HRESULT Cancel( );

Return Values

Returns VFW_E_ALREADY_CANCELLED if m_pQueue is NULL. Returns an HRESULT from CCmdQueue::Remove if the call generates an error. Returns S_OK if successful.

Remarks

This member function implements the IDeferredCommand::Cancel method.

Previous Home Topic Contents Index Next

CDeferredCommand::CDeferredCommand

CDeferredCommand Class

Constructs a CDeferredCommand object.

CDeferredCommand(
  CCmdQueue * *pQ*,
  LPUNKNOWN *pUnk*,
  HRESULT * *phr*,
  LPUNKNOWN *pUnkExecutor*,
  REFTIME *time*,
  GUID* *iid*,
  long *dispidMethod*,
  short *wFlags*,
  long *cArgs*,
  VARIANT* *pDispParams*,
  VARIANT* *pvarResult*,
  short* *puArgErr*,
  BOOL *bStream*
  );

Parameters

pQ
    Object that exposes the IQueueCommand interface.
pUnk
    Outer IUnknown interface for aggregation.
phr
    Returning HRESULT value.
pUnkExecutor
    Object that will carry out this command.
time
    Time at which the command will be run.
iid
    Globally unique identifier (GUID) of the interface that contains the method.
dispidMethod
    Method on the interface to call.
wFlags
    Context of the invocation.

cArgs
  Number of arguments passed.
pDispParams
  List of argument variant types.
pvarResult
  Returned variant type list, if any.
puArgErr
  Last argument in the *pDispParams* parameter list with an error.
bStream
  TRUE if the deferred command time is in stream time; FALSE if in presentation time.

Return Values

No return value.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CDeferredCommand::Confidence

CDeferredCommand Class

This method is not currently implemented.

HRESULT Confidence(
  LONG **pConfidence*
  );

Parameters

pConfidence
  Confidence level.

Return Values

Returns E_NOTIMPL.

Remarks

See IDeferredCommand::Confidence for information about implementing this method.

Previous Home Topic Contents Index Next

CDeferredCommand::GetFlags

CDeferredCommand Class

Returns the context flags associated with the deferred command.

short GetFlags( );

Return Values

The value retrieved will be one of the following.

Value	**Description**
DISPATCH_METHOD	Run the member as a method. If a property has the same name, both this and the DISPATCH_PROPERTYGET flag may be set.
DISPATCH_PROPERTYGET	The member is being retrieved as a property or data member.
DISPATCH_PROPERTYPUT	The member is being changed as a property or data member.
DISPATCH_PROPERTYPUTREF	The member is being changed via a reference assignment, rather than a value assignment. This flag is valid only when the property accepts a reference to an object.

Previous Home Topic Contents Index Next

CDeferredCommand::GetHResult

CDeferredCommand Class

Returns the HRESULT value from the invoked command.

HRESULT GetHResult(
 HRESULT* *phrResult*
 );

Parameters

phrResult
HRESULT value.

Return Values

Returns E_ABORT if m_pQueue is NULL. Otherwise, returns S_OK.

Remarks

This member function implements the IDeferredCommand::GetHResult method.

Previous Home Topic Contents Index Next

CDeferredCommand::GetIID

CDeferredCommand Class

Retrieves the interface identifier (IID) of the interface on which the method will be run.

REFIID GetIID( );

Previous Home Topic Contents Index Next

CDeferredCommand::GetMethod

CDeferredCommand Class

Retrieves the dispatch identifier of the method to be run.

long GetMethod( );

Previous Home Topic Contents Index Next

CDeferredCommand::GetParams

CDeferredCommand Class

Retrieves the DISPPARAMS argument list to the method.

DISPPARAMS* GetParams( );

Previous Home Topic Contents Index Next

CDeferredCommand::GetResult

CDeferredCommand Class

Retrieves the resulting argument list, if one exists.

VARIANT* GetResult( );

Previous Home Topic Contents Index Next

CDeferredCommand::GetTime

CDeferredCommand Class

Returns the time at which the method will be run.

CRefTime GetTime( );

Return Values

Returns a CRefTime object containing a reference time.

Previous Home Topic Contents Index Next

CDeferredCommand::Invoke

CDeferredCommand Class

Provides access to methods and properties exposed by an object.

HRESULT Invoke( );

Return Values

Returns VFW_E_ALREADY_CANCELLED if m_pQueue is NULL. Otherwise, returns the HRESULT resulting from a call to IDispatch::GetTypeInfo or IUnknown::QueryInterface.

Previous Home Topic Contents Index Next

CDeferredCommand::IsStreamTime

CDeferredCommand Class

Specifies whether the command is to be run at stream time or presentation time.

BOOL IsStreamTime( );

Return Values

Returns TRUE if set to stream time; otherwise, returns FALSE.

Previous Home Topic Contents Index Next

CDeferredCommand::NonDelegatingQueryInterfa

CDeferredCommand Class

Returns a specified reference-counted interface.

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

Returns pointers to the IDeferredCommand and IUnknown interfaces by default. Override this method to publish any additional interfaces implemented by the derived class.

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method.

CDeferredCommand::Postpone

CDeferredCommand Class

Specifies a new presentation time for a previously queued command.

HRESULT Postpone(
  REFTIME *newtime*
  );

Parameters

newtime
: New presentation time.

Return Values

Returns VFW_E_TIME_ALREADY_PASSED if *newtime* is already passed. Otherwise, returns an HRESULT resulting from a call to CCmdQueue::Remove (when extracting from the list) or CCmdQueue::Insert (when reinserting with the changed time).

Remarks

This member function implements the IDeferredCommand::Postpone method.

Previous Home Topic Contents Index Next

CDisp Class

CDispBasic
CDisp

The **CDisp** class provides methods for displaying a number of data types for debugging. It provides a constructor for each type, and can be cast to the LPCTSTR type for use as a string in a debug statement.

For example, the following code fragment:

```
int MyFunc(REFERENCE_TIME rt, IPin *pPin)
{
    DbgLog((LOG_TRACE, 2, TEXT("MyFunc(%s, %s)"),
         (LPCTSTR) CDisp(CRefTime(rt)),
         (LPCTSTR) CDisp(pPin)));
...
}
```

could output the following reference time and pin information onto the debug log:

```
Quartz.dll(tid d7) : MyFunc(1.003 sec,
    CLSID_AudioRender(Audio Input pin (rendered)))
```

Member Functions

Name	Description
CDisp	Constructs a CDisp object.

Operators

Name	Description
LPCTSTR	Casts to an LPCTSTR type for use in a debug string.

Previous Home Topic Contents Index Next

CDisp::CDisp

CDisp Class

Constructs a CDisp object.

CDisp(
 LONGLONG *ll*,
 int *Format* **= CDISP_HEX**
 );
CDisp(
 REFCLSID *clsid*
 );
CDisp(
 double *d*
 );
CDisp(
 CRefTime *t*
 );
CDisp(
 IPin **pPin*
 );

Parameters

ll
: LONGLONG value for display.

Format
: Whether the value should be displayed in decimal (CDISP_DEC) or (by default) hexadecimal (CDISP_HEX).

clsid
: Class identifier to display.

d
: The double value to display.

t
: Reference time to display. Note that passing a value of type REFERENCE_TIME will use the LONGLONG constructor.

pPin
: IPin interface to display a pin as "CLSID of the filter(Pin name)"; for example, CLSID_AudioRenderer(Audio Input Pin).

Return Values

No return value.

Remarks

Various constructors are provided, which allows information to be displayed in the most suitable way.

Previous Home Topic Contents Index Next

CDisp::LPCTSTR

CDisp Class

Casts the CDisp object to an LPCTSTR value for use in a debug string.

```
operator LPCTSTR();
```

Return Values

Returns the string representation of the variable used in the constructor.

CDispBasic

CDispBasic

An internal class used to implement the CDisp class.

Member Functions

CDispBasic	Constructor for the CDispBasic class.

Protected Data Members

m_PString	Points to the string to be displayed. Initially, this points to m_String, but larger strings may cause this to be updated to point to dynamically-allocated storage instead.
m_String [50]	The initial buffer area for this object.

CDispBasic::CDispBasic

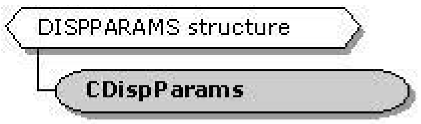
CDispBasic

Instantiates an object of this class.

CDispBasic( );

Previous Home Topic Contents Index Next

CDispParams Class



The **CDispParams** class implements the DISPPARAMS structure used in Automation as a C++ base class. The IDispatch::Invoke method uses the OLE **DISPPARAMS** structure to contain the arguments passed to any method or property.

The DISPPARAMS structure is defined as follows:

```
typedef struct FARSTRUCT tagDISPPARAMS{
VARIANTARG FAR* rgvarg;               // Array of arguments
   DISPID FAR* rgdispidNamedArgs;     // Dispatch IDs of named arguments
   unsigned int cArgs;                // Number of arguments
   unsigned int cNamedArgs;           // Number of named arguments
} DISPPARAMS;
```

Member Functions

Name	**Description**
CDispParams	Constructs a CDispParams object.

Previous Home Topic Contents Index Next

CDispParams::CDispParams

CDispParams Class

Constructs a CDispParams object.

CDispParams(
  UINT *nArgs*,
  VARIANT* *pArgs*
  );

Parameters

nArgs
  Number of arguments passed to the method or property.
pArgs
  Pointer to the list of arguments. In the list, each argument is stored with its variant type.

Return Values

No return value.

CDrawImage Class

CDrawImage

This class is a worker class for the owning CBaseWindow object. It handles the actual drawing operation from that class. To use this class, be sure to call CDrawImage::NotifyAllocator when the allocator has been agreed upon, and call CDrawImage::NotifyMediaType with a pointer to a CMediaType object (which must not be stack-based, because a pointer is maintained by this class rather than making a copy) when that is agreed.

When the palette changes, call CDrawImage::IncrementPaletteVersion, and before rendering call CDrawImage::SetDrawContext so that the class can obtain the handle to a device context (HDC) handles from the owning CBaseWindow object.

Protected Data Members

Name	Description
m_bStretch	Flag to stretch the images.
m_bUsingImageAllocator	Flag to determine if samples share DIBSECTION structures.
m_EndSample	End time for the current sample.
m_hdc	Main window device context (DC).
m_MemoryDC	Offscreen draw DC.
m_pBaseWindow	Owning video window object.
m_PaletteVersion	Current palette version token.
m_perfidRenderNow	Moment when returned from draw (for performance logging).
m_perfidRenderTime	Time taken to render an image (for performance logging).
m_pMediaType	Pointer to the current media type format.
m_SourceRect	Source image rectangle.
m_StartSample	Start time for the current sample.
m_TargetRect	Destination rectangle.

Member Functions

Name	Description
CDrawImage	Constructs a CDrawImage object.
DisplaySampleTimes	Displays a time stamp of a sample on top of its image.
DrawImage	Looks after drawing an image to a window.
FastRender	Draws an image using BitBlt and StretchBlt.
GetPaletteVersion	Retrieves the currently installed palette version.
GetSourceRect	Retrieves the current source rectangle.
GetTargetRect	Retrieves the current target rectangle.
IncrementPaletteVersion	Increments the current palette version.
NotifyAllocator	Notifies the draw object which allocator is being used.

NotifyEndDraw	Indicates the conclusion of image rendering.
NotifyMediaType	Passes the media type established during connection.
NotifyStartDraw	Indicates the beginning of image rendering.
ResetPaletteVersion	Resets the current palette version.
ScaleSourceRect	Returns a scaled version of a provided source rectangle.
SetDrawContext	Sets the window and offscreen device contexts to draw with.
SetSourceRect	Sets the source rectangle for the video.
SetStretchMode	Determines whether it is necessary to stretch.
SetTargetRect	Sets the target rectangle for the window.
SlowRender	Uses the Microsoft® Win32® SetDIBitsToDevice and StretchDIBits functions to draw an image.
UpdateColourTable	Updates the palette held in a DIBSECTION structure.
UsingImageAllocator	Retrieves the type of samples to be drawn.

Previous Home Topic Contents Index Next

Previous Home Topic Contents Index Next

CDrawImage::CDrawImage

CDrawImage Class

Constructs a CDrawImage object.

CDrawImage(
 CBaseWindow **pBaseWindow*
 );

Parameters

pBaseWindow
 Window where drawing will occur.

Return Values

No return value.

Remarks

This class handles drawing of images through GDI. It works closely in conjunction with the CImageAllocator and CBaseWindow classes. It must know about the **CImageAllocator** class, because the draw code provides a faster drawing implementation if the buffers it is handed are created through the Microsoft® Win32® CreateDIBSection function. The image allocator creates this type of sample. It is told whether the buffers are allocated by a **CImageAllocator** object (or derived class) via the CDrawImage::NotifyAllocator member function.

If the buffers used to draw are not allocated by a compatible allocator, it will draw using the Win32 SetDIBitsToDevice family of APIs. The CBaseWindow class retrieves the window handle where the images are to be drawn. The device contexts that the drawing code should use are passed in through the CDrawImage::SetDrawContext member function.

The CImageAllocator, CImageSample, and CDrawImage classes are all tightly associated. The buffers that the image allocator creates are made using the Win32 CreateDIBSection function. The allocator then creates its own samples (based on the **CImageSample** class). The image samples are initialized with the buffer pointer and its length. The sample is also passed in a structure (a DIBDATA structure) that holds a number of pieces of information obtained from the **CreateDIBSection** function.

These samples can then be passed to the draw object. The draw object knows the private format of the samples, and how to get the DIBDATA structure back from them. Once the draw object has obtained that information, it can pass a bitmap handle, stored in the **DIBDATA** structure, down into GDI when it draws the image that the sample contains. By using the bitmap handle from the sample in the drawing, rather than just the buffer pointer (which is the alternative if the sample is not a CImageSample), it gets a modest performance improvement.

Previous Home Topic Contents Index Next

CDrawImage::DisplaySampleTimes

CDrawImage Class

Displays time stamp of a sample on top of the image.

void DisplaySampleTimes(
  IMediaSample **pSample*
  );

Parameters

pSample
Sample containing time stamps.

Return Values

No return value.

Remarks

In debugging builds, it is often instructive to see the time stamps for images that the object is drawing. This member function gets the data pointer for the image the sample holds, along with its time stamps; then, using an offscreen device context, it draws the times approximately 80 percent of the way down the image (and centered horizontally).

This is a protected member function.

Previous Home Topic Contents Index Next

CDrawImage::DrawImage

CDrawImage Class

Entry point for drawing an image.

BOOL DrawImage(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
  Sample to draw.

Return Values

No return value.

Remarks

If the samples have been allocated by a CImageAllocator object (or a derived class), the images that the samples contain will be drawn using the Microsoft Win32 BitBlt or StretchBlt function. If not, they will be drawn using SetDIBitsToDevice or StretchDIBits. The client must call CDrawImage::NotifyAllocator prior to calling this member function to inform the CDrawImage object how the image buffers have been allocated. The object is informed each time the source or destination changes (through its CDrawImage::SetSourceRect and CDrawImage::SetTargetRect member functions). It uses this information to determine if it needs to stretch the image during the draw.

Previous Home Topic Contents Index Next

CDrawImage::FastRender

CDrawImage Class

Draws the sample image using the Microsoft Win32 BitBlt and StretchBlt functions.

void FastRender(
 IMediaSample **pMediaSample*
 );

Parameters

pMediaSample
  Sample to draw.

Return Values

No return value.

Remarks

This protected member function is called by CDrawImage with a sample that contains an image buffer. The image buffer must have been allocated through the Win32 CreateDIBSection function and by a CImageAllocator object (or derived class). There are some performance benefits from drawing images created through this mechanism.

Previous Home Topic Contents Index Next

CDrawImage::GetPaletteVersion

CDrawImage Class

Retrieves the current palette version.

LONG GetPaletteVersion( );

Return Values

Returns the palette version.

Remarks

This member function is applicable only when using samples allocated through a CImageAllocator (or derived class) object. For more information about working with palettes, see the CDrawImage::UpdateColourTable member function.

CDrawImage::GetSourceRect

CDrawImage Class

Retrieves the current source rectangle the draw object is using.

void GetSourceRect(
  RECT **pSourceRect*
  );

Parameters

pSourceRect
    Holds the source rectangle.

Return Values

No return value.

CDrawImage::GetTargetRect

CDrawImage Class

Retrieves the current destination rectangle the draw object is using.

void GetTargetRect(
  RECT **pTargetRect*
  );

Parameters

pTargetRect
    Holds the target rectangle.

Return Values

No return value.

Previous Home Topic Contents Index Next

CDrawImage::IncrementPaletteVersion

CDrawImage Class

Increments the current palette version.

void IncrementPaletteVersion( );

Return Values

No return value.

Remarks

This member function is applicable only when using samples allocated through a CImageAllocator (or derived class) object. For more information about working with palettes, see the CDrawImage::UpdateColourTable member function.

Previous Home Topic Contents Index Next

CDrawImage::NotifyAllocator

CDrawImage Class

Notifies the draw object which allocator the output pin is actually going to use.

void NotifyAllocator(
  BOOL *bUsingImageAllocator*
  );

Parameters

bUsingImageAllocator
Flag to indicate whether to use a CImageAllocator object allocator or not.

Return Values

No return value.

Remarks

This member function tells the draw object whose allocator to use. This should be called with TRUE if the filter agrees to use an allocator based around the DirectShow™ CImageAllocator base class. These image buffers are made through CreateDIBSection. Otherwise this should be called with FALSE, and the images will be drawn using SetDIBitsToDevice and StretchDIBits.

Previous Home Topic Contents Index Next

CDrawImage::NotifyEndDraw

CDrawImage Class

Indicates the conclusion of image rendering.

void NotifyEndDraw(void);

Return Values

No return value.

Remarks

This member function is used for performance measurements and just calls the MSR_STOP macro.

Previous Home Topic Contents Index Next

CDrawImage::NotifyMediaType

CDrawImage Class

Provides the image format for the draw object.

void NotifyMediaType(
  CMediaType **pMediaType*
  );

Parameters

pMediaType
    Media type.

Return Values

No return value.

Remarks

The draw object must know the format of the images it will be drawing. For the most part, this is so it can retrieve the palette when the images are 8-bit palettized. A filter using the draw class will usually call this just after completing a connection.

The method does not take a copy of the media type but just stores a pointer (for performance reasons). Therefore, the caller should ensure that the media type is not destroyed inadvertently.

CDrawImage::NotifyStartDraw

CDrawImage Class

Indicates the beginning of image rendering.

void NotifyStartDraw(void);

Return Values

No return value.

Remarks

This member function is used for performance measurements and just calls the MSR_START macro.

CDrawImage::ResetPaletteVersion

CDrawImage Class

Resets the current palette version.

void ResetPaletteVersion( );

Return Values

No return value.

Remarks

This member function is applicable only when using samples allocated through a CImageAllocator (or derived class) object. For more information about working with palettes, see the CDrawImage::UpdateColourTable member function.

CDrawImage::ScaleSourceRect

CDrawImage Class

Returns a scaled version of a provided source rectangle.

virtual RECT ScaleSourceRect(
  const RECT **pSource*
  );

Parameters

pSource
Unscaled source rectangle.

Return Values

Returns the scaled source rectangle (returns unscaled *pSource* by default).

Remarks

The base class implementation does not scale the source rectangle. Derived classes can override this to implement scaling, if required.

Previous Home Topic Contents Index Next

CDrawImage::SetDrawContext

CDrawImage Class

Sets the device contexts used for drawing.

void SetDrawContext( );

Return Values

No return value.

Remarks

The draw object always needs a device context for the window to draw images in. It might also need an offscreen device context to select bitmaps into when using DIBSECTION buffers (for more details on these and CreateDIBSection, see the Microsoft Platform SDK documentation). This member function will typically be called by a filter using this class, once it has initialized a window.

Previous Home Topic Contents Index Next

CDrawImage::SetSourceRect

CDrawImage Class

Sets the source rectangle for the video.

void SetSourceRect(
  RECT **pSourceRect*
  );

Parameters

pSourceRect
New source rectangle.

Return Values

No return value.

Remarks

The source rectangle should already have been validated before calling this member function so that the source rectangle specified will not extend over the edges of the available video.

CDrawImage::SetStretchMode

CDrawImage Class

Decides whether the video is to be stretched.

void SetStretchMode( );

Return Values

No return value.

Remarks

When the object is asked to draw an image, the object must know whether the video is being stretched, because it affects the function it calls (BitBlt or StretchBlt, for example). Rather than calculate this for every frame, it works it out just once when the source or destination rectangle is updated. This member function is called by SetSourceRect and SetTargetRect to manage this calculation.

This is a protected member function.

CDrawImage::SetTargetRect

CDrawImage Class

Sets the target rectangle for the video.

void SetTargetRect(
  RECT **pTargetRect*
  );

Parameters

pTargetRect
  New target area.

Return Values

No return value.

Remarks

The destination rectangle should already have been validated before calling this member function, so that the destination specified will not define an empty playback area.

CDrawImage::SlowRender

CDrawImage Class

Draws the sample image using SetDIBitsToDevice and StretchDIBits.

void SlowRender(
  IMediaSample **pMediaSample*
  );

Parameters

pMediaSample
  Sample to draw.

Return Values

No return value.

Remarks

The sample provided should contain the image to draw and should match the format as specified to the draw object through NotifyMediaType.

This is a protected member function.

Previous Home Topic Contents Index Next

CDrawImage::UpdateColourTable

CDrawImage Class

Updates the palette associated with a sample.

void UpdateColourTable(
  HDC *hdc*,
  BITMAPINFOHEADER **pbmi*
  );

Parameters

hdc
  Device context containing the sample image.
pbmi
  BITMAPINFO structure containing the new palette.

Return Values

No return value.

Remarks

This member function is applicable only when using samples allocated through a CImageAllocator (or derived class) object. **CImageAllocator** creates samples that are created with the Microsoft Win32 CreateDIBSection function. When a palettized buffer is allocated through **CreateDIBSection**, a palette is passed in that is associated with that buffer.

Should the palette be changed, the new palette must be associated with the buffer before drawing it (this is done through the Win32 SetDIBColorTable function and internally with the **UpdateColourTable** member function). The drawing code knows to update the palette because the palette version it stores in the sample will differ from the palette version it keeps internally.

In essence, the sample gets an initial palette version when created. When the palette is changed (probably by a filter), it tells the draw object to increment its palette version (through the IncrementPaletteVersion member function). When the draw object next comes to draw the sample, it will see that the sample has an old palette version and will know to call **UpdateColourTable** on it.

The draw object knows the type of buffer used for samples through the NotifyAllocator member function. If it is called with TRUE, the buffers passed to it must be allocated by a CImageAllocator (or derived class) object. If it is called with FALSE, the buffers should be allocated in standard system memory (or other memory accessible to GDI in the same manner).

When the allocator is decommitted, it will typically delete all the samples it holds on to. When it is subsequently committed, the samples will be created again with their initial palette versions. At this point, the allocator should also reset the palette version in the draw object so that they remain in sync. An allocator can do this by calling the ResetPaletteVersion member function.

This is a protected member function.

Previous Home Topic Contents Index Next

CDrawImage::UsingImageAllocator

CDrawImage Class

Retrieves the type of samples to be drawn.

BOOL UsingImageAllocator( );

Return Values

Returns one of the following values.

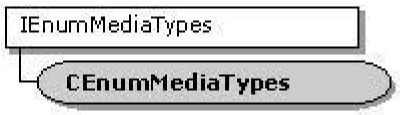
Value	Meaning
TRUE	Allocated through CreateDIBSection.
FALSE	Not allocated through CreateDIBSection.

Remarks

This member function is applicable only when using samples allocated through a CImageAllocator (or derived class) object. For more information about working with palettes and the image allocator, see the CDrawImage::UpdateColourTable member function.

Previous Home Topic Contents Index Next

CEnumMediaTypes Class



This class provides the mechanism for enumerating the pin's preferred media types. Its constructor must be passed to an object from a class derived from CBasePin. It uses the virtual member function GetMediaType to retrieve each of the media types in turn. It also uses the pin's CBasePin::GetMediaTypeVersion member function to determine if the number or type of media types has changed.

The base pin class does not support dynamic media type changes. CBaseFilter::GetPinVersion always returns the same value, for example.

The media type enumerator must fill in a list of pointers to media type structures. The memory for those media type structures must be released by the callers when they have finished with it. However, the memory must not be allocated from any language-specific heap; otherwise, problems might occur between a filter written in C and another written in C++. For this reason, the base classes provide generic functions (not member functions of a class) to create and delete media types: CreateMediaType and DeleteMediaType. These manage memory allocation from the task allocator.

All member functions in this class that return HRESULT and accept a pointer as a parameter return E_POINTER when passed a null pointer.

Member Functions

Name	Description
CEnumMediaTypes	Constructs a CEnumMediaTypes object.

Implemented IUnknown Methods

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

Implemented IEnumMediaTypes Methods

Name	Description
Clone	Creates a duplicate CEnumMediaTypes object with the same state.
Next	Returns the next media type after the current position.
Reset	Sets the current position back to the beginning.
Skip	Skips over one or more entries in the enumerator.

Previous Home Topic Contents Index Next

CEnumMediaTypes::AddRef

CEnumMediaTypes Class

Increments the reference count for the calling interface on an object. It should be called for every new copy of a pointer to an interface on a given object.

ULONG AddRef(void);

Return Values

Returns an integer from 1 to *n*, the value of the new reference count. This information is meant to be used for diagnostic/testing purposes only, because, in certain situations, the value might be unstable.

Remarks

This member function implements the IUnknown::AddRef method.

CEnumMediaTypes::CEnumMediaTypes

CEnumMediaTypes Class

Constructs a CEnumMediaTypes object.

CEnumMediaTypes(
  CBasePin **pPin*,
  CEnumMediaTypes **pEnumMediaTypes*
  );

Parameters

pPin
Pointer to the pin on which the enumeration is to be performed.

pEnumMediaTypes
 Pointer to the instantiated CEnumMediaTypes object.

Return Values

No return value.

Remarks

This is a standard class constructor.

Previous Home Topic Contents Index Next

CEnumMediaTypes::Clone

CEnumMediaTypes Class

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

HRESULT Clone(
 IEnumMediaTypes ** *ppEnum*
 );

Parameters

ppEnum
 New copy of the enumerator.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IEnumMediaTypes::Clone method.

Previous Home Topic Contents Index Next

CEnumMediaTypes::Next

CEnumMediaTypes Class

Retrieves the specified number of items in the enumeration sequence.

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

Parameters

cMediaTypes
    Number of media types to place.
ppMediaTypes
    Array in which to place the next media type or types.
pcFetched
    Actual count passed.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IEnumMediaTypes::Next method. To call this method, pass a pointer's address to a media type. The base class implementation relies on the existence of an overridden CBasePin::GetMediaType member function in the derived class that will provide the next media type.

Free each media type acquired by calling DeleteMediaType, which will free the format block and the media type itself.

CEnumMediaTypes::QueryInterface

CEnumMediaTypes Class

Retrieves a pointer to a specified interface on a component to which a client currently holds an interface pointer.

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

Parameters

iid
  Specifies the IID of the interface being requested.
ppvObject
  Receives a pointer to an interface pointer to the object on return. If the interface specified in *iid* is not supported by the object, *ppvObject* is set to NULL.

Return Values

Returns S_OK if the interface is supported, S_FALSE if not.

Remarks

This member function implements the IUnknown::QueryInterface method and passes out references to the IEnumMediaTypes interface. Override this class to return other interfaces on the object in the derived class.

CEnumMediaTypes::Release

CEnumMediaTypes Class

Decrements the reference count for the calling interface on an object. If the reference count on the object falls to zero, the object is freed from memory.

ULONG Release(void);

Return Values

Returns the resulting value of the reference count, which is used for diagnostic/testing purposes only. If you need to know that resources have been freed, use an interface with higher-level semantics.

Remarks

This member function implements the IUnknown::Release method.

CEnumMediaTypes::Reset

CEnumMediaTypes Class

Resets the enumerator to the beginning so that the next call to the IEnumMediaTypes::Next method will return, at a minimum, the first media type in the enumeration.

HRESULT Reset(void);

Return Values

Returns S_OK if successful; otherwise, returns S_FALSE.

Remarks

This member function implements the IEnumMediaTypes::Reset method.

CEnumMediaTypes::Skip

CEnumMediaTypes Class

Skips a specified number of elements in the enumeration sequence.

HRESULT Skip(
  ULONG *cMediaTypes*
  );

Parameters

cMediaTypes
Number of media type elements to skip.

Return Values

Returns an HRESULT value.

Remarks

This member function implements the IEnumMediaTypes::Skip method.

CEnumPins Class

IEnumPins
CEnumPins

This class supports the IEnumPins enumeration interface by calling CBaseFilter methods. The **CBaseFilter** class supports the IBaseFilter::EnumPins method. Each time one of this interface's methods is called, the **CBaseFilter** class checks to see if the pins that it enumerates have changed; it does this by calling CBaseFilter::GetPinVersion and matching the version the filter is keeping with the version that it stores during construction.

If a pin enumerator becomes stale, there is no mechanism for resynchronizing it with the filter. The user must release the interface and retrieve another one.

Because the enumeration operation is likely to fail if the pin version changes (indicating that the filter might have added or removed pins), all member functions in this class check the version by calling a private member function, which calls the owning filter's CBaseFilter::GetPinVersion member function. These member functions then return VFW_E_ENUM_OUT_OF_SYNC if the version has changed. This should always work unless the filter has overridden **CBaseFilter::GetPinVersion** to do something unexpected.

All member functions in this class that return HRESULT and accept a pointer as a parameter return E_POINTER when passed a null pointer.

Member Functions

Name	Description
CEnumPins	Constructs a CEnumPins object.

Implemented IUnknown Methods

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

Implemented IEnumPins Methods

Name	Description
Clone	Creates a duplicate CEnumPins object with the same initial state.
Next	Returns the next pin after the current position.
Reset	Sets the current position back to the beginning.
Skip	Skips over one or more entries in the enumerator.

Previous Home Topic Contents Index Next

CEnumPins::AddRef

CEnumPins Class

Increments the reference count for the calling interface on an object.

ULONG AddRef(void);

Return Values

Returns an integer from 1 to *n*, the value of the new reference count.

Remarks

This member function implements the IUnknown::AddRef method.

CEnumPins::CEnumPins

CEnumPins Class

Constructor for the CEnumPins class.

CEnumPins(
  CBaseFilter **pFilter*,
  CEnumPins **pEnumPins*
  );

Parameters

pFilter
Pointer to the filter on which to enumerate the pins.
pEnumPins
Returned pointer to an IEnumPins interface object.