

## Return Values

No return value.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CEnumPins::Clone

[CEnumPins Class](#)

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

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

## Parameters

*ppEnum*  
New enumerator that is a copy of this enumerator.

## Return Values

Returns an [HRESULT](#) value.

## Remarks

This member function implements the [IEnumPins::Clone](#) method.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CEnumPins::Next

[CEnumPins Class](#)

Places pointers to [IPin](#) interfaces into the specified array.

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

### Parameters

*cPins*

Number of pins to place.

*ppPins*

Array in which to place the interface pointers.

*pcFetched*

Actual number of pins placed in the array.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IEnumPins::Next](#) method. The derived class is responsible for implementing [CBaseFilter::GetPin](#), which this member function calls to retrieve the next pin.

Because this member function returns one or more interfaces that have had their reference counts incremented, the caller of this member function must be sure to release the interfaces by calling [IUnknown::Release](#) on the interfaces when done with them.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CEnumPins::QueryInterface

[CEnumPins Class](#)

Retrieves a pointer to a specified interface on a component to which a client currently holds an interface pointer. This method must call [IUnknown::AddRef](#) on the pointer it returns.

```
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 [IEnumPins](#) interface.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CEnumPins::Release

[CEnumPins 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.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CEnumPins::Reset

### CEnumPins Class

Resets the enumerator to the beginning so that the next call to the [IEnumPins::Next](#) method will return, at a minimum, the first pin of the filter.

**HRESULT Reset(void);**

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IEnumPins::Reset](#) method.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CEnumPins::Skip

### CEnumPins Class

Skips the next specified number of pins.

**HRESULT Skip(  
    **ULONG** *cPins*  
);**

### Parameters

*cPins*  
    Number of pins to skip.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IEnumPins::Skip](#) method. This member function affects the next call to the [IEnumPins::Next](#) method.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).



## CFactoryTemplate Class

### CFactoryTemplate

This class provides a template used by the default class factory code.

Create one **CFactoryTemplate** object in an array for every object class so that the default class factory code can create new instances.

This class holds the name of the object, the object's class identifier (CLSID), and a pointer to the creation function for the corresponding object. Initialize one of these in an array called `g_Templates` for each CLSID the application's dynamic-link library (DLL) supports. The creation function should take an LPUNKNOWN parameter and an [HRESULT](#) pointer and return an object derived from the [CBaseObject](#) class. Set the **HRESULT** to a failed value if there is any error in construction. An example declaration (from the Gargle sample filter) follows:

```
// list of class ids and creator functions for class factory
CFactoryTemplate g_Templates[2] = {
    { L"Gargle filter"           // CFactoryTemplate.r
    , &CLSID_Gargle             // CFactoryTemplate.r
    , OGargle::CreateInstance   // CFactoryTemplate.r
    , NULL
    , &sudGargle
    }
    , { L"Gargle filter property page"
    , &CLSID_GargProp
    , OGargleProperties::CreateInstance
    }
};

int g_cTemplates = sizeof(g_Templates) / sizeof(g_Templates[0]);
```

Note that the name of the object is strictly necessary only if you are using the [DllRegisterServer](#) setup routine to implement self-registering of your filter. If you are not using this feature, you can set the first element of the `g_Templates` instance (`m_Name`) to NULL or L<sup>""</sup>.

### Protected Data Members

Name	Description
<code>m_ClsID</code>	Pointer to the CLSID of the object class.
<code>m_lpfNew</code>	Pointer to a function that creates an instance of the object class.
<code>m_lpfInit</code>	Pointer to a function that initializes a new instance of the object class.
<code>m_Name</code>	Name of the filter; required when using filter self-registration services.
<code>m_pAMovieSetup_Filter</code>	Pointer to an <a href="#">AMOVIESETUP_FILTER</a> structure; required when using filter self-registration services.

## Member Functions

Name	Description
------	-------------

<a href="#">CreateInstance</a>	Calls the object-creation function for the class.
--------------------------------	---

<a href="#">IsClassID</a>	Determines whether a CLSID matches this class template.
---------------------------	---

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CFactoryTemplate::CreateInstance

## [CFactoryTemplate Class](#)

Calls the object-creation function for the class.

```
CUnknown *CreateInstance(  
    LPUNKNOWN pUnk,  
    HRESULT *phr  
);
```

### Parameters

*pUnk*

Pointer to the [IUnknown](#) interface.

*phr*

Pointer to the [HRESULT](#) value into which to place resulting information.

### Return Values

Returns an instance of the class object.

### Remarks

The implementer of the class code registered using this factory template class is responsible for providing the code that creates an instance of the class object and assigning it to the [m\\_lpfnNew](#) data member. This member function simply calls that function and returns a new object of that type.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CFactoryTemplate::IsClassID

### CFactoryTemplate Class

Determines if the class identifier (CLSID) passed matches the CLSID assigned to this class template.

```
BOOL IsClassID(  
    REFCLSID rclsid  
);
```

### Parameters

*rclsid*  
CLSID being tested.

### Return Values

Returns TRUE if the CLSIDs are the same; otherwise, returns FALSE.

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



[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## CGenericList Class



**CGenericList** is a template class that allows for a type-specific implementation of a list. It is derived from **CBaseList** and uses that class's typeless implementation. The constructor creates a **CBaseList** object, and all **CGenericList** member functions call **CBaseList** member functions but provide type-checking dependent on the template.

### Member Functions

Name	Description
<a href="#"><u>AddAfter</u></a>	Inserts a node or list of nodes after the specified node.
<a href="#"><u>AddBefore</u></a>	Inserts a node or list of nodes before the specified node.
<a href="#"><u>AddHead</u></a>	Inserts a node or list of nodes at the front of the list.
<a href="#"><u>AddTail</u></a>	Appends a node or list of nodes to the end of the list.
<a href="#"><u>CGenericList</u></a>	Constructs a <b>CGenericList</b> object.
<a href="#"><u>Find</u></a>	Returns the first position that contains the specified object.
<a href="#"><u>Get</u></a>	Returns the object at the specified position.
<a href="#"><u>GetCount</u></a>	Returns the number of objects (object count) in the list.
<a href="#"><u>GetHead</u></a>	Returns the object at the head of the list.
<a href="#"><u>GetHeadPosition</u></a>	Returns a cursor identifying the first element of the list.
<a href="#"><u>GetNext</u></a>	Returns the specified object and update position.
<a href="#"><u>GetTailPosition</u></a>	Returns a cursor identifying the last element of the list.
<a href="#"><u>Remove</u></a>	Removes the specified node from the list.
<a href="#"><u>RemoveHead</u></a>	Removes the first node in the list.
<a href="#"><u>RemoveTail</u></a>	Removes the last node in the list.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::AddAfter

CGenericList Class

Inserts a node or list of nodes after the specified node.

```
POSITION AddAfter(
    POSITION p,
    OBJECT * pObj
);
BOOL AddAfter(
    POSITION pos,
    CGenericList<OBJECT> *pList
);
```

**Parameters**

*pos*  
Position after which to add the node or list of nodes.

*pObj*  
Pointer to the object to add.

*pList*  
Pointer to the list of objects to add.

**Return Values**

Returns the position of the inserted object in the case of single-object insertion. For list insertion, returns TRUE if successful; otherwise, returns FALSE.

**Remarks**

This member function calls the [CBaseList::AddAfter](#) member function when passed a list of nodes. **CGenericList::AddAfter** calls the [CBaseList::AddAfterI](#) member function when passed a single node.

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CGenericList::AddBefore

CGenericList Class

Inserts a node or list of nodes before the specified node.

```
POSITION AddBefore(
    POSITION p,
    OBJECT * pObj
);
```

```
);
BOOL AddBefore(
    POSITION pos,
    CGenericList<OBJECT> *pList
);
```

### Parameters

*pos*

Position before which to add the node or list of nodes.

*pObj*

Pointer to the object to add.

*pList*

Pointer to the list of objects to add.

### Return Values

Returns the position of the inserted object in the case of single-object insertion. For list insertion, returns TRUE if successful; otherwise, returns FALSE.

### Remarks

This member function calls the [CBaseList::AddBefore](#) member function when passed a list of nodes. **CGenericList::AddBefore** calls the [CBaseList::AddBeforeI](#) member function when passed a single node.

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CGenericList::AddHead

[CGenericList Class](#)

Inserts a node or list of nodes at the front of the list.

```
POSITION AddHead(
    OBJECT * pObj
);
BOOL AddHead(
    CGenericList<OBJECT> *pList
);
```

### Parameters

*pObj*

Pointer to the object to add.

*pList*

Pointer to the list of objects to add.

### Return Values

Returns the new head position, or NULL if unsuccessful in the case of single-node additions. For list insertions, returns TRUE if successful; otherwise, returns FALSE.

### Remarks

This member function calls the [CBaseList::AddHead](#) member function when passed a list of nodes. **CGenericList::AddHead** calls the [CBaseList::AddHeadI](#) member function when passed a single node.

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

[Previous](#)

[Home](#)

[Topic Contents](#)

[Index](#)

[Next](#)

---

## CGenericList::AddTail

### [CGenericList Class](#)

Appends a node or list of nodes to the end of the list.

```
POSITION AddTail(
    OBJECT * pObj
);
BOOL AddTail(
    CGenericList<OBJECT> *pList
);
```

### Parameters

*pObj*

Pointer to the object to add.

*pList*

Pointer to the list of objects to add.

### Return Values

Returns the new tail position, or NULL if unsuccessful in the case of single-node insertions. For list insertions, returns TRUE if successful; otherwise, returns FALSE.

### Remarks

This member function calls the [CBaseList::AddTail](#) member function when passed a list of nodes. **CGenericList::AddTail** calls the [CBaseList::AddTailI](#) member function when passed a

single node.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::CGenericList

### CGenericList Class

Constructs a CGenericList object.

```
CGenericList(  
    TCHAR *pName,  
    INT iItems,  
    BOOL bLock,  
    BOOL bAlert  
);
```

```
CGenericList(  
    TCHAR *pName  
);
```

### Parameters

*pName*

Name of the list.

*iItems*

Number of items in the list.

*bLock*

TRUE if the list is locked and FALSE otherwise. This parameter defaults to TRUE.

*bAlert*

Not used.

### Return Values

No return value.

### Remarks

This constructor calls the CBaseList constructor.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::Find

### CGenericList Class

Retrieves the first position that contains the specified object.

```
POSITION Find(  
    OBJECT * pObj  
);
```

#### Parameters

*pObj*  
Pointer to the object to find.

#### Return Values

Returns a position cursor.

#### Remarks

This member function calls the CBaseList::FindI member function.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::Get

### CGenericList Class

Retrieves the object at the specified position.

```
OBJECT *Get(  
    POSITION pos  
);
```

#### Parameters

*pos*  
Position in the list from which to retrieve the object.

#### Return Values

Returns a pointer to an object.

### Remarks

This member function calls the [CBaseList::GetI](#) member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::GetCount

[CGenericList Class](#)

Retrieves the number of objects (object count) in the list.

**int GetCount( );**

### Return Values

Returns the value of [m\\_Count](#).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::GetHead

[CGenericList Class](#)

Retrieves the object at the head of the list.

**OBJECT GetHead( );**

### Return Values

Returns the head of the list by calling [CGenericList::GetHeadPosition](#).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::GetHeadPosition

### CGenericList Class

Retrieves a cursor identifying the first element of the list.

**POSITION GetHeadPosition( );**

### Return Values

Returns the position cursor held by m\_pFirst.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next ▶](#)

---

## CGenericList::GetNext

### CGenericList Class

Retrieves the specified object and update position.

**OBJECT \*GetNext(  
POSITION& *rp*  
);**

### Parameters

*rp*  
Returned pointer to the next object.

### Return Values

Returns a pointer to an object at the next position.

### Remarks

This member function calls the CBaseList::GetNextI member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next ▶](#)



---

## CGenericList::GetTailPosition

### CGenericList Class

Retrieves a cursor identifying the last element of the list.

**POSITION** GetTailPosition( );

### Return Values

Returns the position cursor held by m\_pLast.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::Remove

### CGenericList Class

Removes the specified node from the list.

**OBJECT \*Remove(**  
**POSITION** *pos*  
**);**

### Parameters

*pos*  
Position in the list of nodes to remove.

### Return Values

Returns the pointer to the object that was removed.

### Remarks

This member function calls the CBaseList::RemoveI member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::RemoveHead

### CGenericList Class

Removes the first node in the list.

**OBJECT \*RemoveHead( );**

### **Return Values**

Returns the pointer to the object that was removed.

### **Remarks**

This member function calls the CBaseList::RemoveHeadI member function.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CGenericList::RemoveTail

### CGenericList Class

Removes the last node in the list.

**OBJECT \*RemoveTail( );**

### **Return Values**

Returns the pointer to the object that was removed.

### **Remarks**

This member function calls the CBaseList::RemoveTailI member function.

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

[Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next](#)

## CGuidNameList Class

### CGuidNameList

This class implements an array of globally unique identifier (**GUID**) names based on the predefined names of **GUIDs** that come with Microsoft® DirectShow™. (This might or might not include user-defined **GUIDs**.) To get the name used for a **GUID**, look it up in the [GuidNames](#) array:

```
int MyFunc (AM_MEDIA_TYPE mt)
{
    DbgLog ((LOG_TRACE, 2, TEXT("MyFunc: Type %s, Subtype %s"),
            GuidNames [mt.majorType],
            GuidNames [mt.subtype]
            ));
    ...
}
```

#### Operators

Name	Description
------	-------------

<a href="#">operator[ ]</a>	Allows access to the <a href="#">GUID</a> name for a given <b>GUID</b> .
-----------------------------	--

#### Global Data

Name	Description
------	-------------

<a href="#">GuidNames</a>	Array of <a href="#">CGuidNameList</a> objects describing the predefined names of <b>GUIDs</b> that come with DirectShow. (This might or might not include user-defined <b>GUIDs</b> .)
---------------------------	---

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next](#)

---

## CGuidNameList::operator[ ]

### [CGuidNameList](#) Class

Allows access to the [GUID](#) name for a given **GUID**.

```
TCHAR *operator[ ](
    const GUID& guid
);
```

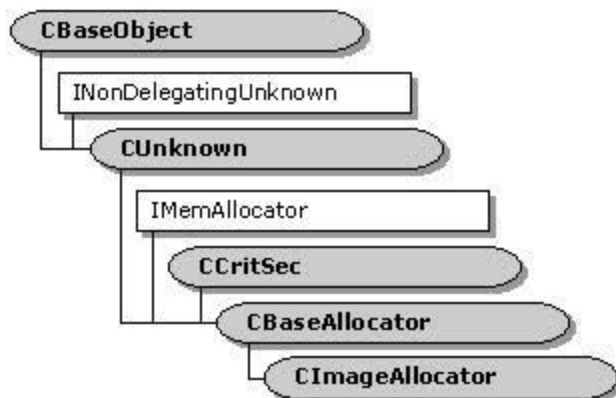
**Parameters***guid*

Globally unique identifier.

**Return Values**Returns the GUID name for the given entry in a **GUID** name list.

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

## CImageAllocator Class



The **CImageAllocator** class is inherited from the **CBaseAllocator** class, which allocates sample buffers in shared memory. The number, size, and alignment of blocks are determined when the connected output pin calls **CImageAllocator::SetProperties** (which implements **IMemAllocator::SetProperties**). The shared memory blocks are used in subsequent calls to the Microsoft® Win32® **CreateDIBSection** function. The output pin can then fill these buffers with data, and the buffers will be handed to GDI using **BitBlt**.

### Protected Data Members

Name	Description
<b>m_pFilter</b>	Owning filter of this object.
<b>m_pMediaType</b>	Current media type format.

### Member Functions

Name	Description
<b>Alloc</b>	Allocates the samples through <b>CreateDIBSection</b> .
<b>CheckSizes</b>	Checks the allocator requirements.
<b>CImageAllocator</b>	Constructs a <b>CImageAllocator</b> object.
<b>CreateDIB</b>	Creates a device-independent bitmap (DIB).
<b>Free</b>	Releases and deletes the resources for any samples allocated.
<b>NotifyMediaType</b>	Notifies the allocator of the agreed media type.

### Overridable Member Functions

Name	Description
<b>CreateImageSample</b>	Creates a sample.

### Implemented INonDelegatingUnknown Methods

Name	Description
<a href="#">NonDelegatingAddRef</a>	Increments the reference count for an interface.
<a href="#">NonDelegatingRelease</a>	Decrements the reference count for an interface.

### Implemented IMemAllocator Methods

Name	Description
<a href="#">SetProperties</a>	Specifies the buffering requirements for the allocator.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageAllocator::Alloc

[CImageAllocator Class](#)

Creates image samples based around [CreateDIBSection](#).

**HRESULT Alloc( );**

### Return Values

Returns an [HRESULT](#) value.

### Remarks

A filter defines the size and number of buffers required through the [CImageAllocator::SetProperties](#) member function. The base allocator class that this allocator derives from calls this internal virtual member function when it wants the memory actually committed. For each sample it wants to create, this allocator will create a [DIBSECTION](#) object for it (through the Microsoft Win32 [CreateDIBSection](#) function). With the information it gets from that call, it will call the virtual [CreateImageSample](#) member function, passing in the buffer pointer and length. After successfully creating an image sample, it will then initialize it with the **DIBSECTION** structure, among other information.

This is a protected member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageAllocator::CheckSizes

### CImageAllocator Class

Internal member function that checks the required buffering properties.

```
HRESULT CheckSizes(  
    ALLOCATOR_PROPERTIES *pRequest  
);
```

### Parameters

*pRequest*  
Requested allocator properties.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

The image allocator uses the Microsoft Win32 [CreateDIBSection](#) function to allocate its samples. That function accepts as input a pointer to a [BITMAPINFO](#) structure that describes the bitmap required. Because the size of the bitmap is therefore fixed according to the **BITMAPINFO** structure for the video, requests to the allocator for a buffer larger than that will not be granted. This member function, therefore, adjusts the requested size so that it is no larger than the size of the bitmap. If the requested size is smaller than the bitmap size, it returns `E_INVALIDARG`.

This is a protected member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageAllocator::CImageAllocator

### CImageAllocator Class

Constructs a [CImageAllocator](#) object.

```
CImageAllocator(  
    CBaseFilter *pFilter,  
    TCHAR *pName,  
    HRESULT *p hr  
);
```

## Parameters

*pFilter*

Owning filter object.

*pName*

Debug-only string description.

*phr*

COM return code.

## Return Values

No return value.

## Remarks

The CImageAllocator, CImageSample, and CDrawImage classes are all tightly associated. The buffers that the image allocator creates are made using the Microsoft 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** call.

These samples can then be passed to the draw object. The draw object knows the private format of the samples and how to get back the DIBDATA structure from them. Once it has obtained that information, it can pass a bitmap handle that is 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.

This is a protected member function.

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

# CImageAllocator::CreateDIB

CImageAllocator Class

Calls the Win32 CreateDIBSection function to create a device-independent bitmap (DIB).

```
HRESULT CreateDIB(  
    LONG InSize,  
    DIBDATA &DibData  
);
```



## Parameters

### *InSize*

Size of the bitmap required.

### *DibData*

Structure to fill out with details.

## Return Values

Returns an [HRESULT](#) value.

## Remarks

This is a protected member function.

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

# CImageAllocator::CreateImageSample

## CImageAllocator Class

Creates a [CImageSample](#) object.

```
virtual CImageSample *CreateImageSample(  
    LPBYTE pData,  
    LONG Length  
);
```

## Parameters

### *pData*

Pointer to the data buffer the sample looks after.

### *Length*

Associated length of the buffer.

## Return Values

Returns a new [CImageSample](#) sample object.

## Remarks

This virtual member function creates the actual sample for the allocator. It is passed the data buffer and its length to store. When the sample is subsequently asked for the buffer (through [IMediaSample::GetPointer](#)), this is the pointer it will return. The primary reason for having this split out into a separate virtual member function is so that derived classes from

[CImageAllocator](#) can also derive classes from [CImageSample](#) and have a place to create them.

This is a protected member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageAllocator::Free

[CImageAllocator Class](#)

Deletes the samples and frees their resources.

**void Free( );**

### Return Values

No return value.

### Remarks

The base allocator calls this internal virtual member function when it wants to decommit the allocator.

This is a protected member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageAllocator::NonDelegatingAddRef

[CImageAllocator Class](#)

Increments the reference count for the owning filter.

**HRESULT NonDelegatingAddRef( );**

### Return Values

Returns an [HRESULT](#) value.

## Remarks

An allocator is conceptually a separate object from the filter that creates it. However, the image allocator is dependent on the filter that created it to supply it with additional information (such as the media type that it connected with). Therefore, although the allocator looks after its own **NonDelegatingQueryInterface** function, it delegates all reference counting to the owning filter. So, when the allocator is subject to its **NonDelegatingAddRef** function, for example, it is the filter that owns the allocator that will actually be reference counted.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CImageAllocator::NonDelegatingRelease

## CImageAllocator Class

Decrements the reference count for the owning filter.

**HRESULT NonDelegatingRelease( );**

## Return Values

Returns an HRESULT value.

## Remarks

An allocator is conceptually a separate object from the filter that creates it. However, the image allocator is dependent on the filter that created it to supply it with additional information (such as the media type that it connected with). Therefore, although the allocator looks after its own **NonDelegatingQueryInterface** function, it delegates all reference counting to the owning filter. So when the allocator is released, for example, it is the filter that owns the allocator that will actually be released by the **NonDelegatingRelease** function.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CImageAllocator::NotifyMediaType

## CImageAllocator Class

Passes the media type from a filter to the allocator.

```
void NotifyMediaType(  
    CMediaType *pMediaType  
);
```

### Parameters

*pMediaType*  
Media type the filter established.

### Return Values

No return value.

### Remarks

The buffers that the image allocator creates are based around [CreateDIBSection](#), which must be told what sort of bitmap the filter requires it to create. The filter does this by calling this member function on the allocator. A filter will usually call this member function after agreeing on a media type during a pin connection. The media type passed to this member function is a pointer; the allocator stores this pointer (not a copy) of the media type it points to (for performance reasons, copying media types is relatively slow). Therefore, the filter that calls this member function should ensure that the media type is always valid until the media type is next set on the allocator (or is called with a NULL type).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CImageAllocator::SetProperties

### CImageAllocator Class

Determines the size, number, and alignment of blocks.

```
HRESULT SetProperties(  
    ALLOCATOR_PROPERTIES * pRequest,  
    ALLOCATOR_PROPERTIES * pActual  
);
```

### Parameters

*pRequest*  
Requested allocator properties.

*pActual*  
Allocator properties actually set.

### Return Values

Returns an HRESULT value.

### Remarks

The *pRequest* parameter is filled in by the caller with the requested values for the count, number, and alignment as specified by the ALLOCATOR\_PROPERTIES structure. The *pActual* parameter is filled in by the allocator with the closest values that it can provide for the request. This member function cannot be called unless the allocator has been decommitted by using the IMemAllocator::Decommit method.

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

## CImageDisplay Class

CCritSec

CImageDisplay

This class initializes itself with a display format so that other objects can query or reset the display type. It also provides member functions to check display formats and accept only those video formats that can be efficiently rendered by using GDI calls.

### Protected Data Members

Name	Description
------	-------------

<u>m_Display</u>	<u>VIDEOINFOHEADER</u> structure corresponding to the current device display type.
------------------	--

### Member Functions

Name	Description
------	-------------

<u>CheckBitFields</u>	Checks that the bit fields on a <u>VIDEOINFOHEADER</u> structure are correct.
-----------------------	---

<u>CheckHeaderValidity</u>	Determines if a <u>BITMAPINFOHEADER</u> structure is valid.
----------------------------	---

<u>CheckMediaType</u>	Determines if the filter can support the media type proposed by the output pin.
-----------------------	---

<u>CheckPaletteHeader</u>	Determines if the palette on a <u>VIDEOINFOHEADER</u> structure is correct.
---------------------------	---

<u>CheckVideoType</u>	Compares a video type to determine if it is compatible with the current display mode.
-----------------------	---

<u>CImageDisplay</u>	Constructs a <u>CImageDisplay</u> object.
----------------------	---

<u>CountPrefixBits</u>	Counts the number of prefix bits.
------------------------	-----------------------------------

<u>CountSetBits</u>	Counts the total number of bits set in a field.
---------------------	---

<u>GetBitMasks</u>	Retrieves a set of color element bitmasks for the supplied <u>VIDEOINFOHEADER</u> structure.
--------------------	--

<u>GetColourMask</u>	Retrieves a set of individual color element masks.
----------------------	--

<u>GetDisplayDepth</u>	Retrieves the bit depth of the current display mode.
------------------------	--

<u>GetDisplayFormat</u>	Retrieves a <u>VIDEOINFOHEADER</u> structure representing the current display mode.
-------------------------	---

<u>IsPalettized</u>	Determines if the display uses a palette.
---------------------	---

<u>RefreshDisplayType</u>	Updates the <u>CImageDisplay</u> object with the current display type.
---------------------------	--

<u>UpdateFormat</u>	Updates the <u>VIDEOINFOHEADER</u> structure to remove implicit assumptions.
---------------------	--

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CheckBitFields

[CImageDisplay Class](#)

Checks that the bit fields in the [VIDEOINFOHEADER](#) structure are correct.

```
BOOL CheckBitFields(  
    const VIDEOINFO *pInput  
);
```

### Parameters

*pInput*  
[VIDEOINFOHEADER](#) structure to check.

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Bit fields are correct.

FALSE Bit fields contain an error.

### Remarks

The assumption throughout the object is that any bitmasks are allowed no more than 8 bits to store a color component. This member function checks that the bit count assumption is enforced, and also ensures that all the bits set are contiguous.

This is a protected member function.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CheckHeaderValidity

[CImageDisplay Class](#)

Determines if a [BITMAPINFOHEADER](#) structure is valid.

```
BOOL CheckHeaderValidity(  
    const VIDEOINFO *pInput  
);
```

### Parameters

*pInput*  
[VIDEOINFOHEADER](#) structure that contains the bitmap details.

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Format is valid.

FALSE Format contains an error.

### Remarks

The [BITMAPINFOHEADER](#) structure might be rejected for a number of reasons. These might include a number-of-planes entry greater or less than one, the size of the structure not being equal to the size of [BITMAPINFOHEADER](#), or, perhaps, being asked to validate a YUV format (this member function only validates RGB formats; it will always return FALSE for YUV types).

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CheckMediaType

[CImageDisplay Class](#)

Determines if the filter can support the media type proposed by the output pin.

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

### Parameters

*pmtIn*  
Media type to check.



## Return Values

Returns an [HRESULT](#) value.

## Remarks

This helper member function can be used to validate a video media type. It examines the major and minor type [GUIDs](#) and verifies that the format **GUID** defines a [VIDEOINFOHEADER](#) structure.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CImageDisplay::CheckPaletteHeader

## [CImageDisplay Class](#)

Determines if the palette on a [VIDEOINFOHEADER](#) structure is correct.

```
BOOL CheckPaletteHeader(  
    const VIDEOINFO *pInput  
);
```

## Parameters

*pInput*  
[VIDEOINFOHEADER](#) structure to validate.

## Return Values

Returns one of the following values.

### Value Meaning

TRUE Palette is correct.

FALSE No valid palette.

## Remarks

This member function returns FALSE if the format specifies that no palette is available (it might be a true-color format). It also returns FALSE if the number of palette colors used (or those that are important) exceeds the number specified for the video format.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CheckVideoType

[CImageDisplay Class](#)

Compares a video type to determine if it is compatible with the current display mode.

```
HRESULT CheckVideoType(  
    const VIDEOINFO *pInput  
);
```

### Parameters

*pInput*  
VIDEOINFOHEADER structure to validate.

### Return Values

Returns NOERROR if successful or E\_INVALIDARG if unsuccessful.

### Remarks

Many video rendering filters want a function to determine if proposed formats are okay. This member function checks the [VIDEOINFOHEADER](#) structure passed as a media type and returns NOERROR if the media type is valid; otherwise, it returns E\_INVALIDARG434. Note, however, that only formats that can be easily displayed on the current display device are accepted; so, for example, a 16-bit device will not accept 24-bit images. Because most displays draw 8-bit palettized images efficiently, this format is always accepted unless the display is 16-color VGA.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CImageDisplay

[CImageDisplay Class](#)

Constructs a [CImageDisplay](#) object.

```
CImageDisplay( );
```

## Return Values

No return value.

## Remarks

The CImageDisplay class helps renderers that want to determine the format of the current display mode. This member function retrieves the display mode and creates a VIDEOINFOHEADER structure that represents its format. The class supplies that format for clients through member functions such as IsPalettized and GetDisplayFormat. If a client detects the display format has changed (perhaps it receives a WM\_DISPLAYCHANGED message), it should call RefreshDisplayType.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CImageDisplay::CountPrefixBits

## CImageDisplay Class

Helper member function to count the number of prefix bits.

```
DWORD CountPrefixBits(  
    const DWORD Field  
);
```

## Parameters

*Field*  
Input bitmask field.

## Return Values

No return value.

## Remarks

Given a bitmask, this member function counts the number of zero bits up to the least significant set bit. So, for a binary number 00000100, the member function returns 2 (decimal). The member function does, however, work on DWORD values, so it counts from the least significant bit up through the **DWORD** to the last bit (0x80000000). If no bits are found, this will return the (impossible) value 32 (decimal).

This is a protected member function.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::CountSetBits

[CImageDisplay Class](#)

Counts the number of bit sets in the *Field* parameter.

```
DWORD CountSetBits(  
    const DWORD Field  
);
```

### Parameters

*Field*  
Field in which to count bit sets.

### Return Values

Returns the number of bit sets.

### Remarks

This is a protected member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::GetBitMasks

[CImageDisplay Class](#)

Retrieves a set of color element bitmasks for the supplied [VIDEOINFOHEADER](#) structure.

```
const DWORD *GetBitMasks(  
    const VIDEOINFO *pVideoInfo  
);
```

### Parameters

*pVideoInfo*

Input [VIDEOINFOHEADER](#) structure format.

### Return Values

No return value.

### Remarks

This member function should be called only with RGB formats. If the RGB format has a bit depth of 16/32 bits per pixel, it will return the bitmasks for the individual red, green, and blue color elements (for example, RGB565 is 0xF800, 0x07E0, and 0x001F). For RGB24, this will return 0xFF0000, 0xFF00, and 0xFF. For palettized formats, this will return all zeros.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::GetColourMask

[CImageDisplay Class](#)

Retrieves a set of individual color element masks.

```
BOOL GetColourMask(  
    DWORD *pMaskRed,  
    DWORD *pMaskGreen,  
    DWORD *pMaskBlue  
);
```

### Parameters

*pMaskRed*  
Holds red mask.

*pMaskGreen*  
Holds green mask.

*pMaskBlue*  
Holds blue mask.

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Masks were filled out correctly.

FALSE No masks were available for the display.

### Remarks

Given a video format described by a [VIDEOINFOHEADER](#) structure, this member function returns the mask that is used to obtain the range of acceptable colors for this type (for example, the mask for a 24-bit true color format is 0xFF in all cases). A 16-bit 5:6:5 display format uses 0xF8, 0xFC, and 0xF8. Therefore, given any RGB triplets, this member function can find one that is compatible with the display format by using a bitwise-AND operation.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CImageDisplay::GetDisplayDepth

[CImageDisplay Class](#)

Retrieves the bit depth of the current display mode.

**WORD GetDisplayDepth( );**

### Return Values

Returns the number of bits per pixel used on the display.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CImageDisplay::GetDisplayFormat

[CImageDisplay Class](#)

Retrieves a [VIDEOINFOHEADER](#) structure representing the current display mode.

**const VIDEOINFO \*GetDisplayFormat( );**

### Return Values

Returns a [VIDEOINFOHEADER](#) structure representing the display format.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::IsPalettized

[CImageDisplay Class](#)

Determines if the display uses a palette.

**BOOL IsPalettized( );**

### Return Values

Returns TRUE if the display uses a palette; otherwise, returns FALSE.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::RefreshDisplayType

[CImageDisplay Class](#)

Updates the [CImageDisplay](#) object with the current display type.

**HRESULT RefreshDisplayType(  
LPSTR szDeviceName  
);**

### Parameters

*szDeviceName*

LPSTR value that contains the name of the device to update. If omitted, this parameter defaults to the main device.

### Return Values

Returns NOERROR if successful; E\_FAIL if unsuccessful.

### Remarks

This member function should be called when a WM\_DISPLAYCHANGED message is received.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageDisplay::UpdateFormat

### CImageDisplay Class

Updates the VIDEOINFOHEADER structure to remove implicit assumptions.

```
HRESULT UpdateFormat(  
    VIDEOINFO *pVideoInfo  
);
```

### Parameters

*pVideoInfo*  
VIDEOINFOHEADER structure to update.

### Return Values

Returns an HRESULT value. Current implementation returns NOERROR.

### Remarks

This member function is probably suitable only for specific filters to use. The BITMAPINFO structure has certain fields that are not well specified. In particular, the number of colors specified for a palette can be zero, in which case it is defined to be the maximum for that format type. This member function updates these fields so that their contents are explicit.

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



## CImagePalette Class

### CImagePalette

The **CImagePalette** class is a specialized class for image renderers that must create and manage palettes. It can be used to create palette handles from a media format containing a **VIDEOINFO** structure in the format block. To maximize performance, the class attempts to create a palette that is an identity palette (that is, one that exactly matches the current system palette), and compares palettes before updating to ensure that palettes are changed only when actually required.

#### Protected Data Members

Name	Description
<b>m_hPalette</b>	Palette handle owned by this object.
<b>m_pBaseWindow</b>	Window in which to realize the palette.
<b>m_pDrawImage</b>	Object that will perform the drawing.
<b>m_pMediaFilter</b>	Media filter to send events to.

#### Member Functions

Name	Description
<b>CImagePalette</b>	Constructs a <b>CImagePalette</b> object.
<b>CopyPalette</b>	Copies the palette out of any YUV or true-color <b>VIDEOINFOHEADER</b> structure into a palettized <b>VIDEOINFOHEADER</b> structure.
<b>MakeIdentityPalette</b>	Ensures the palette entries will become an identity palette.
<b>MakePalette</b>	Retrieves the color palette from the specified video image.
<b>PreparePalette</b>	Specifies an entry point for updating and creating palettes.
<b>RemovePalette</b>	Releases any palette resources allocated.
<b>ShouldUpdate</b>	Specifies an internal helper member function for updating palettes dynamically.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

---

## CImagePalette::CImagePalette

## CImagePalette Class

Constructs a CImagePalette object.

```
CImagePalette(  
  CBaseFilter *pBaseFilter,  
  CBaseWindow *pBaseWindow,  
  CDrawImage *pDrawImage  
);
```

### Parameters

*pBaseFilter*

Filter that this class is owned by.

*pBaseWindow*

Window to realize palette in.

*pDrawImage*

Object that draws using this palette.

### Return Values

No return value.

### Remarks

This class looks after the creation, management, and deletion of a window palette. It is passed in a number of other objects that might be interested in palettes. The class is optimized so that requested palette changes will be acted on only if the new set of colors differs from the current set. This is a performance optimization, because changing palettes is an expensive process.

This constructor is passed in the owning filter (*pBaseFilter*), which must be a valid pointer. When the class actually creates a palette, it tells the owning filter to send an `EC_PALETTE_CHANGED` message to the filter graph manager. The constructor might also be passed two further object pointers. If *pBaseWindow* is not null, when the renderer creates a new palette the class automatically installs it in this window. When told to remove a palette, the class also removes the palette from the base window and installs a standard VGA palette instead.

The constructor can also be passed a drawing object derived from the CDrawImage class. If this is non-NULL, when creating a new palette the class will inform the drawing object that the palette has changed (this is usually used in conjunction with a window object). This ensures that the drawing object is notified when the palette changes so that it can update any samples it has that were created using CreateDIBSection (because they might need their internal color tables updated).

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CImagePalette::CopyPalette

### CImagePalette Class

Copies the palette out of any YUV or true-color VIDEOINFOHEADER structure into a palettized **VIDEOINFOHEADER** structure.

```
HRESULT CopyPalette(
    const CMediaType *pSrc,
    const CMediaType *pDest
);
```

### Parameters

*pSrc*  
Source media type.

*pDest*  
Destination media type.

### Return Values

Returns NOERROR if successful or S\_FALSE if no palette is available.

### Remarks

This member function is used when changing palettes on DirectDraw® samples. A filter acting as a source to the renderer can attach a palette to any buffer and pass it to the renderer as a new VIDEOINFOHEADER format. The renderer can then call **CopyPalette** to make a new palette from that format, and copy the palette colors into the new connection type.

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

[Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next](#)

---

## CImagePalette::MakeIdentityPalette

### CImagePalette Class

Modifies the PALETTEENTRY structure to create an identity palette.

```
HRESULT MakeIdentityPalette(
    PALETTEENTRY *pEntry,
    INT iColours,
    LPSTR szDevice
);
```

## Parameters

*pEntry*

Array of prospective palette colors.

*iColours*

Number of colors in the array.

*szDevice*

LPSTR value that contains the name of the destination device. If omitted, this parameter defaults to the main device.

## Return Values

Returns NOERROR if successful or S\_FALSE if unsuccessful.

## Remarks

When a palette is installed in a window, GDI does a fair job of compressing the requested colors where possible. So, for example, if the array contains five entries of black, they will be compressed into one palette entry. This is useful for most applications; however, when drawing video it will force GDI to map the pixels in the supplied image to the compressed palette (which results in serious performance penalties).

Therefore, the PALETTEENTRY fields supplied must be adjusted so that they will never have colors compressed. This means that when the window displaying the image has the foreground focus, the palette created by this object will map directly to the palette selected in the display device: a so-called identity palette.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CImagePalette::MakePalette

CImagePalette Class

Retrieves the color palette from the specified video image.

```
HPALETTE MakePalette(
  const VIDEOINFOHEADER *pVideoInfo,
  LPSTR szDevice
);
```

## Parameters

*pVideoInfo*

Container for the palette colors required.

*szDevice*

LPSTR value that contains the name of the destination device. If omitted, this parameter defaults to the main device.

### Return Values

Returns a handle to the new palette (NULL if it fails).

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CImagePalette::PreparePalette

### CImagePalette Class

Specifies an entry point for creating and installing palettes.

```
HRESULT PreparePalette(  
    const CMediaType *pmtNew,  
    const CMediaType *pmtOld,  
    LPSTR szDevice  
);
```

### Parameters

*pmtNew*

Media type holding new palette information.

*pmtOld*

Media type holding old palette information.

*szDevice*

LPSTR value that contains the name of the destination device. If omitted, this parameter defaults to the main device.

### Return Values

Returns an HRESULT value.

### Remarks

This is the main entry point for creating new palettes. It tries to detect situations where the palette colors requested have not changed (in which case it does not need to create a new palette). It uses the old media type to determine if the colors have changed. It also handles optionally installing the palette in a window (if supplied) and notifying the filter graph manager of a change in palettes (it uses the filter passed in to the constructor for this). Finally, it handles notifying the draw object of palette changes (also optional, depending on whether a draw object was passed in to the constructor).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImagePalette::RemovePalette

[CImagePalette Class](#)

Removes and deletes any palette previously created.

**HRESULT RemovePalette( );**

### Return Values

Returns an [HRESULT](#) value. Current implementation returns NOERROR.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImagePalette::ShouldUpdate

[CImagePalette Class](#)

[Help](#) member function that checks if two sets of colors match.

```
BOOL ShouldUpdate(  
    const VIDEOINFOHEADER *pNewInfo,  
    const VIDEOINFOHEADER *pOldInfo  
);
```

### Parameters

*pNewInfo*

[VIDEOINFOHEADER](#) structure containing the new set of colors.

*pOldInfo*

[VIDEOINFOHEADER](#) structure containing the old set of colors.

### Return Values

Returns one of the following values.

**Value Meaning**

TRUE A new palette is required.

FALSE The existing palette suffices.

[© 1997 Microsoft Corporation. All rights reserved. Terms of Use.](#)

[◀ Previous](#)

[Home](#)

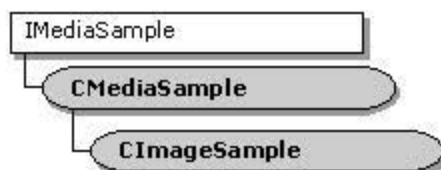
[Topic Contents](#)

[Index](#)

[Next ▶](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## CImageSample Class



This class inherits from [CMediaSample](#) and overrides the constructor to initialize itself with the [DIBDATA](#) structure. When the renderer is using its own allocator, it will use this class for its samples. It can therefore obtain the [DIBSECTION](#) structure information it requires to obtain the [HBITMAP](#) data it renders.

### Protected Data Members

Name	Description
<a href="#">m_bInit</a>	Flag to determine if the <a href="#">DIBSECTION</a> structure information is initialized.
<a href="#">m_DibData</a>	Information about the sample's <a href="#">DIBSECTION</a> structure.

### Member Functions

Name	Description
<a href="#">CImageSample</a>	Constructs a <a href="#">CImageSample</a> object.
<a href="#">GetDIBData</a>	Retrieves the <a href="#">DIBSECTION</a> structure information stored for the sample.
<a href="#">SetDIBData</a>	Sets the <a href="#">DIBSECTION</a> information stored for the sample.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageSample::CImageSample

### [CImageSample Class](#)

Constructs a [CImageSample](#) object.

```

CImageSample(
    CBaseAllocator *pAllocator,
    TCHAR *pName,

```



```
HRESULT *p hr,  
LPBYTE p Buffer,  
LONG length  
);
```

### Parameters

*pAllocator*  
Base allocator to which the sample belongs.

*pName*  
Debug-only string description.

*p hr*  
COM return code.

*pBuffer*  
Pointer to the image buffer.

*length*  
Length of the image buffer.

### Return Values

No return value.

### Remarks

The [CImageAllocator](#), [CImageSample](#), and [CDrawImage](#) classes are all tightly associated. The buffers that the image allocator creates are made by using the Microsoft® 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** call.

These samples can then be passed to the draw object. The draw object knows the private format of the samples and how to get back the [DIBDATA](#) structure from them. Once it has obtained that information, it can pass a bitmap handle, which is 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 achieves a modest performance improvement.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CImageSample::GetDIBData

[CImageSample Class](#)

Retrieves the DIBDATA structure held by the sample.

```
DIBDATA *GetDIBData( );
```

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CImageSample::SetDIBData

CImageSample Class

Sets the DIBDATA structure that the sample should hold.

```
void SetDIBData(  
    DIBDATA *pDibData  
    );
```

### Parameters

*pDibData*  
New DIBDATA structure.

### Return Values

No return value.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## CLoadDirectDraw Class

### CLoadDirectDraw

DirectShow™ must work on multiple platforms; in particular, it also runs on Microsoft® Windows NT® 3.51, which does not have DirectDraw® capabilities. The filters therefore cannot link statically to the DirectDraw library. To make the platform dependencies easier to handle, this class manages loading and unloading the library and creating the initial IDirectDraw interface.

#### Member Functions

Name	Description
<u>CLoadDirectDraw</u>	Constructs a <u>CLoadDirectDraw</u> object.
<u>GetDirectDraw</u>	Retrieves a pointer to the <u>IDirectDraw</u> interface.
<u>IsDirectDrawLoaded</u>	Verifies that DirectDraw is loaded.
<u>IsDirectDrawVersion1</u>	Checks the version of DirectDraw installed on the current system.
<u>LoadDirectDraw</u>	Loads and initializes the DirectDraw library.
<u>ReleaseDirectDraw</u>	Releases the <u>IDirectDraw</u> interface.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CLoadDirectDraw::CLoadDirectDraw

### CLoadDirectDraw Class

Constructs a CLoadDirectDraw object.

**CLoadDirectDraw(void);**

#### Return Values

No return value.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CLoadDirectDraw::GetDirectDraw

[CLoadDirectDraw Class](#)

Retrieves the DirectDraw interface.

**LPDIRECTDRAW GetDirectDraw(void);**

### Return Values

Returns a pointer to the [IDirectDraw](#) interface.

### Remarks

Call [CLoadDirectDraw::LoadDirectDraw](#) before calling this member function and call the [CLoadDirectDraw::ReleaseDirectDraw](#) member function to release the interface when you are done.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CLoadDirectDraw::IsDirectDrawLoaded

[CLoadDirectDraw Class](#)

Verifies that this object loaded DirectDraw.

**HRESULT IsDirectDrawLoaded(void);**

### Return Values

Returns S\_OK if loaded; otherwise, returns S\_FALSE.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CLoadDirectDraw::IsDirectDrawVersion1

### CLoadDirectDraw Class

Checks the version of DirectDraw installed on the current system.

#### **BOOL IsDirectDrawVersion1(void);**

#### **Return Values**

Returns TRUE if the installed version of DirectDraw doesn't support the IDirectDraw2 interface, or FALSE if the m\_pDirectDraw data member is NULL or the installed version of DirectDraw supports **IDirectDraw2**.

#### **Remarks**

The video renderer must know what the installed version of DirectDraw is to perform certain tasks, such as full-screen playback, which the IDirectDraw2 interface supports.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## **CLoadDirectDraw::LoadDirectDraw**

### CLoadDirectDraw Class

Loads and initializes the DirectDraw library in the specified area.

#### **HRESULT LoadDirectDraw( LPSTR *szDevice* );**

#### **Parameters**

*szDevice*

This parameter is optional; if omitted, this method loads DirectDraw to the base drawing area.

#### **Return Values**

Returns S\_OK if DirectDraw loaded correctly or E\_NOINTERFACE otherwise.

#### **Remarks**

DirectDraw is not always available, so applications can't statically link to the library. Therefore, this member function loads the library, gets the function entry point addresses, and calls them to create the driver objects. Call this member function before calling

[CLoadDirectDraw::GetDirectDraw](#) to retrieve the [IDirectDraw](#) interface.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CLoadDirectDraw::ReleaseDirectDraw

[CLoadDirectDraw Class](#)

Releases the [IDirectDraw](#) interface.

**void ReleaseDirectDraw(void);**

### Return Values

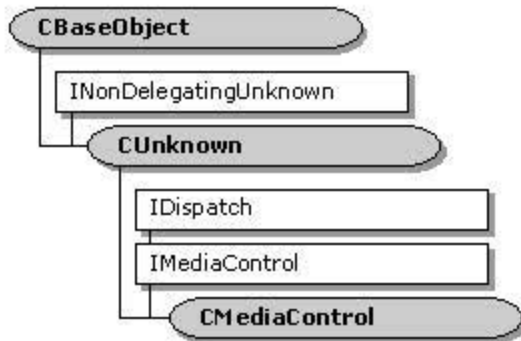
No return value.

### Remarks

This member function is called to release any [IDirectDraw](#) interface previously loaded. Call this only when all reference counts have been released.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

## CMediaControl Class



The **CMediaControl** class provides base class handling of the [IDispatch](#) methods of the dual-interface [IMediaControl](#). It leaves as pure virtual the properties and methods of the **IMediaControl** interface.

Typically, the filter graph manager is the only object that implements the [IMediaControl](#) interface. (Filters implement the [IMediaFilter](#) interface, inherited by [IBaseFilter](#), to receive control commands from the filter graph manager.) Therefore, this class library is of limited use to filter developers.

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

The [IMediaControl](#) methods, defined in [control.odl](#), are left as pure virtual.

### Member Functions

Name	Description
<a href="#">CMediaControl</a>	Constructs a <a href="#">CMediaControl</a> object.

### Implemented [INonDelegatingUnknown](#) Methods

Name	Description
<a href="#">NonDelegatingQueryInterface</a>	Returns a specified reference-counted interface.

### Implemented [IDispatch](#) Methods

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

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

---

## CMediaControl::CMediaControl

### [CMediaControl Class](#)

Constructs a [CMediaControl](#) object.

```
CMediaControl(
    const TCHAR *pName,
    LPUNKNOWN pUnk
);
```

### Parameters

*pName*  
Name of the object 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 the object.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)



---

## CMediaControl::GetIDsOfNames

### CMediaControl Class

Maps a single member function and an optional set of parameters to a corresponding set of integer dispatch identifiers (DISPIDs), which can be used upon subsequent calls to the [CMediaControl::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.

<b>Value</b>	<b>Meaning</b>
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.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaControl::GetTypeInfo

### CMediaControl 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
<code>E_FAIL</code>	Failure.
<code>E_POINTER</code>	Null pointer argument.
<code>E_INVALIDARG</code>	Invalid argument.
<code>S_OK</code> or <code>NOERROR</code>	Success.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaControl::GetTypeInfoCount

### CMediaControl 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.

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

[◀ Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next ▶](#)

---

## CMediaControl::Invoke

### CMediaControl 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 [CMediaControl::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 **CMediaControl::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.

***pexceptinfo***

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 [CMediaControl::GetTypeInfo](#) if the call fails. Otherwise, returns the [HRESULT](#) from the call to [IDispatch::Invoke](#).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaControl::NonDelegatingQueryInterface

**CMediaControl 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.

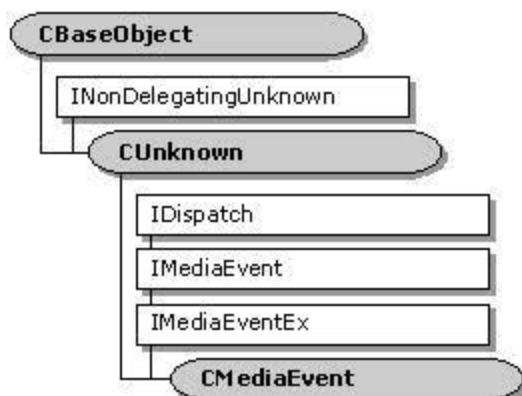
### **Remarks**

Returns pointers to the IMediaControl 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.

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

## CMediaEvent Class



The **CMediaEvent** class provides base class implementation of the [IDispatch](#) methods of the dual-interface [IMediaEvent](#). It leaves as pure virtual the properties and methods of the [IMediaEvent](#) interface.

The **CMediaEvent** class also provides base class implementation of the [IMediaEventEx](#) interface which derives from [IMediaEvent](#).

The [CMediaEvent::GetIDsOfNames](#), [CMediaEvent::GetTypeInfo](#), [CMediaEvent::GetTypeInfoCount](#), and [CMediaEvent::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 methods of the [IMediaEvent](#) interface.

### Member Functions

Name	Description
------	-------------

<a href="#">CMediaEvent</a>	Constructs a <a href="#">CMediaEvent</a> object.
-----------------------------	--

### Implemented [INonDelegatingUnknown](#) Methods

Name	Description
------	-------------

<a href="#">NonDelegatingQueryInterface</a>	Returns a specified reference-counted interface.
---	--

### Implemented [IDispatch](#) Methods

Name	Description
------	-------------

<a href="#">GetIDsOfNames</a>	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 <a href="#">IDispatch::Invoke</a> method.
-------------------------------	---

<a href="#">GetTypeInfo</a>	Retrieves a type-information object, which retrieves the type information for an interface.
-----------------------------	---

<a href="#">GetTypeInfoCount</a>	Retrieves the number of type-information interfaces provided by an object.
----------------------------------	--

[Invoke](#) Provides access to properties and methods exposed by an object.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaEvent::CMediaEvent

### [CMediaEvent Class](#)

Constructs a [CMediaEvent](#) object.

```
CMediaEvent(  
    const TCHAR * pName,  
    LPUNKNOWN pUnk  
);
```

#### Parameters

*pName*

Name of the object 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 the object.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaEvent::GetIDsOfNames

### [CMediaEvent 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 [CMediaEvent::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.

<b>Value</b>	<b>Meaning</b>
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.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## **CMediaEvent::GetTypeInfo**



CMediaEvent 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:

<b>Value</b>	<b>Meaning</b>
<code>E_FAIL</code>	Failure.
<code>E_POINTER</code>	Null pointer argument.
<code>E_INVALIDARG</code>	Invalid argument.
<code>S_OK</code> or <code>NOERROR</code>	Success.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaEvent::GetTypeInfoCount

CMediaEvent 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.

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

[Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next](#)

---

## CMediaEvent::Invoke

CMediaEvent 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 [CMediaEvent::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 **CMediaEvent::Invoke** call.

*pdispparams*

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

*pvarResult*

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

*pexceptinfo*

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 `CMediaEvent::GetTypeInfo` if the call fails. Otherwise, returns the `HRESULT` from the call to `IDispatch::Invoke`.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaEvent::NonDelegatingQueryInterface

[CMediaEvent 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.

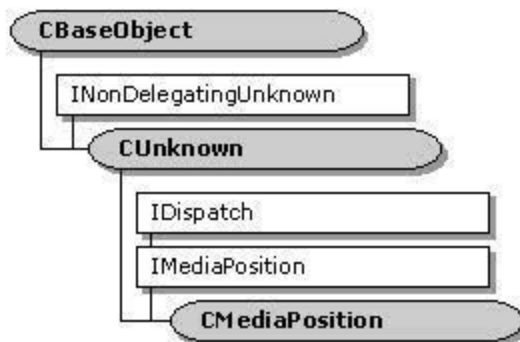
## Remarks

Returns a pointer to the `IMediaEvent` and `IUnknown` interfaces by default. Override this member function to publish any additional interfaces added by the derived class.

This member function implements the INonDelegatingUnknown::NonDelegatingQueryInterface method.

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

## CMediaPosition Class



The **CMediaPosition** class is a base class that handles the [IDispatch](#) methods of the dual-interface [IMediaPosition](#). It leaves the properties and methods as pure virtual.

The [CMediaPosition::GetIDsOfNames](#), [CMediaPosition::GetTypeInfo](#), [CMediaPosition::GetTypeInfoCount](#), and [CMediaPosition::Invoke](#) methods 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 [IMediaPosition](#) methods.

### Member Functions

Name	Description
<a href="#">CMediaPosition</a>	Constructs a <a href="#">CMediaPosition</a> object.

### Implemented INonDelegatingUnknown Methods

Name	Description
<a href="#">NonDelegatingQueryInterface</a>	Returns a specified reference-counted interface.

### Implemented IDispatch Methods

Name	Description
<a href="#">GetIDsOfNames</a>	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 <a href="#">CMediaPosition::Invoke</a> member function.
<a href="#">GetTypeInfo</a>	Retrieves a type-information object, which can retrieve the type information for an interface.
<a href="#">GetTypeInfoCount</a>	Retrieves the number of type-information interfaces provided by an object.
<a href="#">Invoke</a>	Provides access to properties and methods exposed by an object.

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaPosition::CMediaPosition

### CMediaPosition Class

Constructs a CMediaPosition object.

```
CMediaPosition(  
    const TCHAR *pName,  
    LPUNKNOWN pUnk  
);
```

### Parameters

*pName*

Name of the object used in the CMediaPosition 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 debug terminal upon creation and deletion of the object.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaPosition::GetIDsOfNames

### CMediaPosition 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 CMediaPosition::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.

<b>Value</b>	<b>Meaning</b>
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.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next ▶](#)

[◀ Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next ▶](#)

---

## CMediaPosition::GetTypeInfo

CMediaPosition 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
<code>E_FAIL</code>	Failure.
<code>E_POINTER</code>	Null pointer argument.
<code>E_INVALIDARG</code>	Invalid argument.
<code>S_OK</code> or <code>NOERROR</code>	Success.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaPosition::GetTypeInfoCount

[CMediaPosition 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.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaPosition::Invoke

### CMediaPosition 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 * pexceptinfo,
UINT * puArgErr
);
```

**Parameters***dispidMember*

Identifier of the member. Use [CMediaPosition::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 **CMediaPosition::Invoke** call.

*pdispparams*

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

*pvarResult*

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

*pexceptinfo*

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 [CMediaPosition::GetTypeInfo](#) if the call fails. Otherwise, returns the `HRESULT` from the call to [IDispatch::Invoke](#).

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaPosition::NonDelegatingQueryInterface

[CMediaPosition 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 a pointer to [IMediaPosition](#) and [IUnknown](#) interfaces by default. Override this member function to publish any additional interfaces implemented by the derived class.

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

## CMediaSample Class



This class represents a buffer object that supports the [IMediaSample](#) interface. It represents a buffer in memory, together with some associated properties stored as protected data members.

The constructor is passed as a pointer to the buffer with its length in bytes; other properties are normally set and accessed through implemented [IMediaSample](#) interface methods. These properties describe various attributes of the media sample, such as the sample's media type, start and end time, and options. The options can include whether the media sample is a sync point, a preroll sample, or discontinuous with other samples.

All member functions in this class that return [HRESULT](#) and accept a pointer as a parameter return [E\\_POINTER](#) when passed a null pointer.

### Protected Data Members

Name	Description
<b>m_cbBuffer</b>	Size of the buffer.
<b>m_dwFlags</b>	Sample property flags as follows: Sample_Discontinuity: Set if start of a new segment. Sample_MediaTimeValid: Set if the media time is valid. Sample_Preroll: Set if sample is a preroll sample. Sample_StopValid: Set if the stop time is valid. Sample_SyncPoint: Set if sample is a synchronization point. Sample_TimeValid: Set if the time is valid. Sample_TypeChanged: Set if the type has changed.
<b>m_End</b>	Sample end time.
<b>m_lActual</b>	Actual length of data in this sample.
<b>m_MediaEnd</b>	Media end (offset from <a href="#">m_MediaStart</a> ).
<b>m_MediaStart</b>	Media start position.
<b>m_pAllocator</b>	Pointer to the <a href="#">IMemAllocator</a> object associated with this object.
<b>m_pBuffer</b>	Pointer to the complete buffer.
<b>m_pMediaType</b>	Pointer to a structure containing the media type of the sample.
<b>m_pNext</b>	Pointer to the next <a href="#">CMediaSample</a> object in the free list.
<b>m_Start</b>	Sample start time.

### Member Functions

<b>Name</b>	<b>Description</b>
<a href="#">CMediaSample</a>	Constructs a <a href="#">CMediaSample</a> object.
<a href="#">SetPointer</a>	Sets the buffer pointer and length.

### Implemented IUnknown Methods

<b>Name</b>	<b>Description</b>
<a href="#">QueryInterface</a>	Returns pointers to supported interfaces.
<a href="#">AddRef</a>	Increments the reference count.
<a href="#">Release</a>	Decrements the reference count.

### Implemented IMediaSample Methods

<b>Name</b>	<b>Description</b>
<a href="#">GetActualDataLength</a>	Retrieves the data length of the sample.
<a href="#">GetMediaTime</a>	Retrieves the media time extents of the sample.
<a href="#">GetMediaType</a>	Retrieves the media type of the <a href="#">CMediaSample</a> object.
<a href="#">GetPointer</a>	Retrieves a read/write pointer to the memory of this buffer.
<a href="#">GetSize</a>	Returns the size, in bytes, of the buffer data area.
<a href="#">GetTime</a>	Sets the media time extents for this sample.
<a href="#">IsDiscontinuity</a>	Determines if there is discontinuity in the data stream.
<a href="#">IsPreroll</a>	Indicates a preroll property. If TRUE, this sample is for preroll only and should not be displayed.
<a href="#">IsSyncPoint</a>	Determines if the beginning of a sample is a synchronization point.
<a href="#">SetActualDataLength</a>	Sets the data length of the sample.
<a href="#">SetDiscontinuity</a>	Sets the discontinuity property.
<a href="#">SetMediaTime</a>	Sets the media time of the <a href="#">CMediaSample</a> object.
<a href="#">SetMediaType</a>	Sets the media type of the <a href="#">CMediaSample</a> object.
<a href="#">SetPreroll</a>	Sets preroll property. If TRUE, this sample is for preroll only and should not be displayed.
<a href="#">SetSyncPoint</a>	Sets sync-point property.
<a href="#">SetTime</a>	Sets the stream time at which this sample should start and finish.

### Implemented INonDelegatingUnknown Methods

<b>Name</b>	<b>Description</b>
<a href="#">NonDelegatingQueryInterface</a>	Passes out pointers to any interfaces added to the derived filter class.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next ▶](#)

[◀ Previous](#)
[Home](#)
[Topic Contents](#)
[Index](#)
[Next ▶](#)

---

## CMediaSample::AddRef

## CMediaSample 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.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaSample::CMediaSample

## CMediaSample Class

Constructs a CMediaSample object.

```
CMediaSample(  
    TCHAR *pName,  
    CBaseAllocator *pAllocator,  
    HRESULT *phr,  
    LPBYTE pBuffer = NULL,  
    LONG length = 0  
);
```

### Parameters

*pName*

Name of the media sample.

*pAllocator*

Pointer to the CBaseAllocator object used for memory allocation.

*phr*

Pointer to the general COM return value. Note that this value is changed only if this function fails.

*pBuffer*

Pointer to a memory buffer (to be allocated by the *pAllocator* parameter).

*length*

Length of the allocated memory buffer.

### Return Values

No return value.

### Remarks

The constructor creates an object with the buffer and buffer length set to that of the [CBaseAllocator](#) object to which it points.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaSample::GetActualDataLength

### [CMediaSample Class](#)

Retrieves the data length of the sample.

**HRESULT GetActualDataLength(void);**

### Return Values

Returns the value of [m\\_lActual](#) by default.

### Remarks

This member function implements the [IMediaSample::GetActualDataLength](#) method.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaSample::GetMediaTime

### [CMediaSample Class](#)

Retrieves the starting and ending media time.

```
HRESULT GetMediaTime(  
    LONGLONG * pStart,  
    LONGLONG * pEnd  
);
```

### Parameters

*pStart*  
Retrieved beginning media time.

*pEnd*  
Retrieved ending media time.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IMediaSample::GetMediaTime](#) method. It sets *pStart* to the current value of [m\\_MediaStart](#) and *pEnd* to the sum of **m\_MediaStart** and [m\\_MediaEnd](#). If the sample has not been set, this member function returns [VFW\\_E\\_MEDIA\\_TIME\\_NOT\\_SET](#).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CMediaSample::GetMediaType

### [CMediaSample Class](#)

Retrieves the media type of the [CMediaSample](#) object.

```
HRESULT GetMediaType(  
    AM_MEDIA_TYPE ** ppMediaType  
);
```

### Parameters

*ppMediaType*  
Pointer to a pointer to the retrieved media type.

### Return Values

Returns an [HRESULT](#) value. When a sample is received and there is no format change, this method returns [S\\_FALSE](#).

### Remarks

This member function implements the [IMediaSample::GetMediaType](#) method. The member function makes a copy of the [AM\\_MEDIA\\_TYPE](#) structure and creates a task memory block to maintain the reference. When you are done with the media type, free the memory block with the [FreeMediaType](#) utility function, and then free the entire media type with the Microsoft® Win32® [CoTaskMemFree](#) function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::GetPointer

[CMediaSample Class](#)

Retrieves a read/write pointer to the buffer's memory.

```
HRESULT GetPointer(  
    BYTE ** ppBuffer  
);
```

### Parameters

*ppBuffer*  
Retrieved pointer to the buffer.

### Return Values

Returns [VFW\\_E\\_BUFFER\\_NOTSET](#) if [CMediaSample::SetPointer](#) was not called before calling this function, or [NOERROR](#) otherwise.

### Remarks

This member function implements the [IMediaSample::GetPointer](#) method. **GetPointer** returns the value of [m\\_pBuffer](#), set using [CMediaSample::SetPointer](#).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::GetSize

[CMediaSample Class](#)



Retrieves the size, in bytes, of the buffer data area.

**HRESULT GetSize(void);**

### Return Values

Returns the value of `m_cbBuffer` by default.

### Remarks

This member function implements the `IMediaSample::GetSize` method.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::GetTime

### CMediaSample Class

Retrieves the stream time at which this sample should begin and finish.

**HRESULT GetTime(  
REFERENCE\_TIME \* *pTimeStart*,  
REFERENCE\_TIME \* *pTimeEnd*  
);**

### Parameters

*pTimeStart*  
Retrieved beginning stream time.

*pTimeEnd*  
Retrieved ending stream time.

### Return Values

Returns `VFW_E_SAMPLE_TIME_NOT_SET` if this sample doesn't have valid timestamps, or `NOERROR` otherwise.

### Remarks

This member function implements the `IMediaSample::GetTime` method. It sets *pTimeStart* to the current value of `m_Start` and *pTimeEnd* to the current value of `m_End`.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::IsDiscontinuity

### CMediaSample Class

Determines if there is discontinuity in the data stream.

#### **HRESULT IsDiscontinuity(void);**

#### **Return Values**

Returns S\_OK if the sample is a discontinuous sample and S\_FALSE if not; otherwise, returns an [HRESULT](#) error value.

#### **Remarks**

This member function implements the [IMediaSample::IsDiscontinuity](#) method. It returns the value of the [m\\_dwFlags](#) Sample\_Discontinuity property flag. Discontinuity occurs when a source filter seeks to a different place in the stream or when a filter drops samples for quality control.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::IsPreroll

### CMediaSample Class

Preroll property. If TRUE, this sample is for preroll only and should not be displayed.

#### **HRESULT IsPreroll(void);**

#### **Return Values**

Returns S\_OK if the sample is a preroll sample and S\_FALSE if not; otherwise, returns an [HRESULT](#) error value.

#### **Remarks**

This member function implements the [IMediaSample::IsPreroll](#) method. It returns the value of

the `m_dwFlags Sample_Preroll` property flag. Preroll samples are not meant to be rendered.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::IsSyncPoint

### CMediaSample Class

Determines if the beginning of a sample is a synchronization point.

**HRESULT IsSyncPoint(void);**

#### Return Values

Returns `S_OK` if the sample is a synchronization point and `S_FALSE` if not; otherwise, returns an `HRESULT` error value.

#### Remarks

This member function implements the `IMediaSample::IsSyncPoint` method. It returns the value of the `m_dwFlags Sample_SyncPoint` property flag. If the `bTemporalCompression` member of the `AM_MEDIA_TYPE` structure is `FALSE`, all samples are synchronization points. A filter can begin a stream at any synchronization point.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::NonDelegatingQueryInterface

### CMediaSample 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 implements the [INonDelegatingUnknown::NonDelegatingQueryInterface](#) method and passes out references to the [IMediaSample](#) and [IUnknown](#) interfaces. Override this class to return other interfaces on the object in the derived class.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::QueryInterface

### [CMediaSample Class](#)

Retrieves a pointer to a specified interface on a component to which a client currently holds an interface pointer. This method must call [IUnknown::AddRef](#) on the pointer it returns.

```
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.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::Release

### CMediaSample 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.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::SetActualDataLength

### CMediaSample Class

Sets the data length of the sample.

**HRESULT SetActualDataLength(  
    long *lLen*  
);**

### Parameters

*lLen*  
    Length of the data in the media sample, in bytes.

## Return Values

Returns an [HRESULT](#) value.

## Remarks

This member function implements the [IMediaSample::SetActualDataLength](#) method. It sets the value of [m\\_lActual](#) to the value of *lLen*.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaSample::SetDiscontinuity

## [CMediaSample Class](#)

Sets the discontinuity property.

```
HRESULT SetDiscontinuity(  
    BOOL bDiscont  
);
```

## Parameters

*bDiscont*

Set to TRUE to specify the media sample as discontinuous with the previous sample.

## Return Values

Returns an [HRESULT](#) value.

## Remarks

This member function implements the [IMediaSample::SetDiscontinuity](#) method. It sets the value of the [m\\_dwFlags Sample\\_Discontinuity](#) flag to the value of *bDiscont*. Discontinuous samples occur when a source filter seeks to a different place in the media stream or when a filter drops samples for quality control.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaSample::SetMediaTime

## CMediaSample Class

Sets the starting and ending media times.

```
HRESULT SetMediaTime(  
    LONGLONG * pStart,  
    LONGLONG * pEnd  
);
```

### Parameters

*pStart*  
Beginning media time.

*pEnd*  
Ending media time.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IMediaSample::SetMediaTime](#) method. It sets the [m\\_MediaStart](#) data member to the value of *pStart* and the [m\\_MediaEnd](#) data member to the value of *pEnd* minus *pStart*.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaSample::SetMediaType

## CMediaSample Class

Sets the media type for the [CMediaSample](#) object.

```
HRESULT SetMediaType(  
    AM_MEDIA_TYPE * pMediaType  
);
```

### Parameters

*pMediaType*  
Pointer to a pointer to a media type structure to be set.

## Return Values

Returns an [HRESULT](#) value.

## Remarks

This member function implements the [IMediaSample::SetMediaType](#) method. It deletes the previous media type if one exists, makes a copy of the media type passed in, sets [m\\_pMediaType](#) to the copy of the media type, and sets the value of the [m\\_dwFlags](#) [Sample\\_TypeChanged](#) flag to TRUE.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

# CMediaSample::SetPointer

## [CMediaSample Class](#)

Sets the buffer pointer and length.

```
SetPointer(  
    BYTE * ptr,  
    LONG cBytes  
);
```

## Parameters

*ptr*

Pointer to a buffer.

*cBytes*

Length of the buffer, in bytes.

## Return Values

No return value.

## Remarks

Allocators that require variable-sized pointers or pointers into data that has already been read use this member function. This is available only through a [CMediaSample](#) class, not an [IMediaSample](#) interface, so only the filter that owns the allocator knows how to access this member function (not any filter or pin that is passed the object's **IMediaSample** interface pointer).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).



[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::SetPreroll

### CMediaSample Class

Sets the preroll property. If TRUE, this sample is for preroll only and should not be displayed.

```
HRESULT SetPreroll(  
    BOOL bIsPreroll  
);
```

### Parameters

*bIsPreroll*

Set to TRUE to specify the media sample as a preroll sample, or FALSE otherwise.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function implements the [IMediaSample::SetPreroll](#) method. It sets the value of the `m_dwFlags Sample_Preroll` flag to the value of *bIsPreroll*. Preroll samples are samples that are processed but not displayed, and are located in the media stream before the displayable samples.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaSample::SetSyncPoint

### CMediaSample Class

Property of a synchronization point.

```
HRESULT SetSyncPoint(  
    BOOL bIsSyncPoint  
);
```

## Parameters

### *bIsSyncPoint*

Value specifying whether the synchronization point was set.

## Return Values

Returns S\_OK.

## Remarks

This member function implements the IMediaSample::SetSyncPoint method. It sets the value of the m\_dwFlags Sample\_SyncPoint flag to the value of *bIsSyncPoint*. A filter can begin a stream at any synchronization point.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaSample::SetTime

## CMediaSample Class

Sets the media time extents for this sample.

```
HRESULT SetTime(  
    REFERENCE_TIME * pTimeStart,  
    REFERENCE_TIME * pTimeEnd  
);
```

## Parameters

### *pTimeStart*

Stream time at which the sample begins.

### *pTimeEnd*

Stream time at which the sample ends.

## Return Values

Returns NOERROR or an HRESULT value.

## Remarks

This member function implements the IMediaSample::SetTime method. It sets the m\_Start data member to the value of *pTimeStart* and the m\_End data member to the value of *pTimeEnd*.

If *pTimeStart* and *pTimeEnd* are null, DirectShow turns off the m\_dwFlags data member's Sample\_TimeValid and Sample\_StopValid bits.

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

## CMediaType Class

AM\_MEDIA\_TYPE structure

CMediaType

When filters are connected, they typically negotiate a type between them. This type describes the format of the data to be exchanged; if the filters do not agree on a media type, they cannot connect. Microsoft® DirectShow™ describes types through the media type structure, which contains two conceptual parts. The first is a list of members that describes generic attributes of a data stream. An example of this is a member that declares whether the data will be passed in fixed-size buffers. The second part of the structure is a variable-length block of data. How large the block of data should be and what it will contain depend on the type of data stream. For example, if the data stream is digital video, the format block is a [VIDEOINFOHEADER](#) structure. If, on the other hand, it is digital audio, the format block is a Microsoft Win32® [WAVEFORMATEX](#) structure.

A data stream type (for example, digital video) is set with a combination of two globally unique identifiers ([GUIDs](#)), called a major type and a subtype. The *major type* describes the overall class of data, examples of which might be digital video, digital audio, MIDI, or text captions. The *subtype* should supply a more specific description of the data type. In the case of digital video, for example, the subtype could be RGB8, RGB16, or RGB32 (among others). By having these two types in a generic structure ([AM\\_MEDIA\\_TYPE](#)), a component, such as a filter graph, can connect filters without any knowledge that is type specific.

The distinction between what goes in the major type and the subtype is somewhat arbitrary. However, as a general rule, transformations between major types (for example, video to audio or video to MIDI) should be relatively rare. Such a rare exception might be a transformation between audio and MIDI. As for the subtype, the more information promoted from the type-specific format block into the subtype, the better the design.

As an example of promoting type-specific information to the subtype, video in DirectShow uses a [VIDEOINFOHEADER](#) structure for the type-specific format block. This contains a Win32 [BITMAPINFOHEADER](#) structure that defines the video stream. **BITMAPINFOHEADER** contains the bit depth of the video, such as 8-bit, 16-bit, or 24-bit. This information is duplicated in the subtype field, because a subtype of RGB8 directly infers a bit count of 8.

DirectShow defines a number of major types. The most important of these are a video type that uses [VIDEOINFOHEADER](#) for the variable-length format block, and an audio that uses [WAVEFORMATEX](#). However, it is insufficient to have a major type (such as digital video) inferring the contents of the format block (in this case, **VIDEOINFOHEADER**). The principal reason for this is extensibility: the format block type must be able to be updated without changing the less-specific major type. Therefore, what the format block actually contains is inferred by another [GUID](#) called the *format type*. If the format block contains **VIDEOINFOHEADER**, the format type **GUID** will be `FORMAT_VideoInfo`.

The principal use of the **CMediaType** class is to manage a media type structure in a simple way. At the same time, the class provides some extra helper functions (such as format-block copying and allocation). The class can be cast to an AM\_MEDIA\_TYPE structure when an interface method requires one to be passed to it.

The **CMediaType** class contains a pointer to a block of memory. When copying a **CMediaType** object, it is insufficient to simply copy the pointer. In C++, a data copy is required, which actually allocates a new block of memory and copies the data into it. This is the purpose of the copy operator.

Similarly, when comparing two **CMediaType** objects, you must compare the blocks of variable-length data (actually using memcmp) when producing the final result. To make this possible, **CMediaType** overrides the equivalence operator.

### Member Functions

Name	Description
<u>AllocFormatBuffer</u>	Allocates an uninitialized format block in the object.
<u>CMediaType</u>	Constructs a <b>CMediaType</b> object.
<u>Format</u>	Returns the format block for this media type.
<u>FormatLength</u>	Returns the length of the format block of this object.
<u>FormatType</u>	Returns a pointer to the format type.
<u>GetSampleSize</u>	Returns the size of the samples.
<u>InitMediaType</u>	Initializes the media type.
<u>IsFixedSize</u>	Queries whether the samples are fixed in length.
<u>IsPartiallySpecified</u>	Checks if the media type is not completely specified.
<u>IsTemporalCompressed</u>	Queries whether the data stream is compressed temporally.
<u>IsValid</u>	Queries whether the media type is currently valid.
<u>MatchesPartial</u>	Checks whether this media type matches another media type that is only partially specified.
<u>ReallocFormatBuffer</u>	Reallocates the format block, maintaining its current content where possible.
<u>ResetFormatBuffer</u>	Deletes any format block that is currently present.
<u>SetFormat</u>	Sets the format block.
<u>SetFormatType</u>	Sets the type of the format block in the object.
<u>SetSampleSize</u>	Sets the size of the samples.
<u>SetSubtype</u>	Sets the subtype.
<u>SetTemporalCompression</u>	Marks the media type to indicate that samples will be temporally compressed.
<u>SetType</u>	Sets the major type.
<u>SetVariableSize</u>	Marks the media type to indicate that samples will vary in length.
<u>Subtype</u>	Returns a pointer to the subtype.
<u>Type</u>	Returns a pointer to the major type.

### Operators

Name	Description
<code>operator =</code>	Performs a copy operation.
<code>operator ==</code>	Tests for equality between <b>CMediaType</b> objects.
<code>operator !=</code>	Tests for inequality between <b>CMediaType</b> objects.

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::AllocFormatBuffer

[CMediaType Class](#)

Allocates a block of memory for the format block.

```
BYTE* AllocFormatBuffer(  
    ULONG length  
);
```

### Parameters

*length*  
Size required for the format block.

### Return Values

Returns a pointer to the new block if successful; otherwise, returns NULL.

### Remarks

Any previous format block is deleted and a new block is allocated and installed. The size required must be nonzero.

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

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::CMediaType

[CMediaType Class](#)

Constructs a [CMediaType](#) object.

```
CMediaType( );
CMediaType(
    const GUID * majortype
);
CMediaType(
    const AM_MEDIA_TYPE& mtype
);
CMediaType(
    const CMediaType& cmtype
);
```

### Parameters

*majortype*

Major type [GUID](#).

*mtype*

[AM\\_MEDIA\\_TYPE](#) structure.

*cmtype*

[CMediaType](#) object from which this object is constructed.

### Return Values

No return value.

### Remarks

A [CMediaType](#) object can be constructed in a number of different ways. The class provides a default constructor that takes no parameters. It can also be constructed based on an [AM\\_MEDIA\\_TYPE](#) structure or another **CMediaType** object. In both cases, it takes a data copy of the format block before returning.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

---

## CMediaType::Format

[CMediaType Class](#)

Returns a pointer to the variable-length format block of the object.

```
BYTE* Format( ) const;
```

### Return Values

Returns the format block of the object whose content is type-specific.

### Remarks

If no format block has been allocated, it might return NULL.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CMediaType::FormatLength

[CMediaType Class](#)

Returns the size, in bytes, of the format block that the object contains.

**ULONG FormatLength( ) const;**

### Return Values

Returns the length of the format block, or NULL if no format block is present.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CMediaType::FormatType

[CMediaType Class](#)

Retrieves the format type.

**const GUID \*FormatType( ) const;**

### Return Values

Returns a pointer to the format type.

### Remarks



The format [GUID](#) describes the content of the variable-length format block. Examples of format types are `FORMAT_VideoInfo` and `FORMAT_WaveFormatEx`.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::GetSampleSize

[CMediaType Class](#)

Returns the maximum sample size for the data stream.

**ULONG GetSampleSize( ) const;**

### Return Values

Returns the maximum size of any sample to be sent, or zero to indicate that the sample size is variable.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::InitMediaType

[CMediaType Class](#)

Initializes the sample.

**void InitMediaType( );**

### Return Values

No return value.

### Remarks

This member function clears memory, sets the fixed sample size property, and sets the sample size to 1.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::IsFixedSize

### CMediaType Class

Determines if the samples for the stream will be fixed or variable size.

**BOOL IsFixedSize( ) const;**

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Samples will be fixed size.

FALSE Samples will be variable length.

[© 1997 Microsoft Corporation. All rights reserved. Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::IsPartiallySpecified

### CMediaType Class

Determines if the media type is only partially defined. This is the case if the major type or format type is GUID\_NULL.

**BOOL IsPartiallySpecified( ) const;**

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Media type is partially specified.

FALSE Media type is completely specified.

### Remarks

This function does not check the sub type.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::IsTemporalCompressed

### CMediaType Class

Asks if the stream will be compressed temporally.

**BOOL IsTemporalCompressed( ) const;**

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Stream will have temporal compression.

FALSE Stream will have no temporal compression.

### Remarks

Some data streams, such as compressed video, have temporal dependencies between successive samples. Other data streams do not have temporal dependencies between their samples; that is, each sample can be treated as an independent unit; for example, MIDI.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::IsValid

### CMediaType Class

Queries whether the object has a valid major type.

**BOOL IsValid( ) const;**

### Return Values

Returns one of the following values.

**Value Meaning**

TRUE `CMediaType` object has a valid major type.

FALSE `CMediaType` object does not have a valid major type.

**Remarks**

When `CMediaType` objects are constructed, their GUIDs are initialized with `GUID_NULL` (unless they are constructed based on another `AM_MEDIA_TYPE` structure or **`CMediaType`** object). This member function is useful for discovering if the object has been correctly initialized.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::MatchesPartial

`CMediaType` Class

Determines if this media type matches the media type pointed to by the *ppartial* parameter.

```
BOOL MatchesPartial(  
    const CMediaType *ppartial  
    ) const;
```

**Parameters**

*ppartial*  
 Pointer to the media type to match.

**Return Values**

Returns one of the following values.

**Value Meaning**

TRUE Media types match for the parts that are defined.

FALSE Media types do not match.

**Remarks**

The matching applies only for the parts of *ppartial* that are defined. That is, this only matches the major type, subtype, or format type of the media type if these are not defined as `GUID_NULL`.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::ReallocFormatBuffer

### CMediaType Class

Reallocates the format block to a new size.

```
BYTE* ReallocFormatBuffer(  
    ULONG length  
);
```

### Parameters

*length*

New size required for the format block.

### Return Values

Returns a pointer to the new block if successful; otherwise, returns NULL.

### Remarks

Any current format block will be copied into the newly allocated block up to its maximum size. Any excess will be lost when the new block is smaller than the old one. When the new block is larger, the excess is not filled with zeros.

The size required must be nonzero.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::ResetFormatBuffer

### CMediaType Class

Deletes any format block currently held, sets it to NULL, and sets the size of the format block to zero.

```
void ResetFormatBuffer( );
```

## Return Values

No return value.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMediaType::SetFormat

[CMediaType Class](#)

Sets the variable-length format block.

```
BOOL SetFormat(  
    BYTE *pFormat,  
    ULONG length  
);
```

## Parameters

*pFormat*

Block of memory containing type-specific information.

*length*

Overall length of the format block.

## Return Values

Returns one of the following values.

### Value Meaning

TRUE Format block was set.

FALSE An error occurred; most likely there was no memory available.

## Remarks

The function takes a copy of the format block and stores that internally.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## CMediaType::SetFormatType

### CMediaType Class

Sets the GUID that describes the content of the format block.

```
void SetFormatType(  
    const GUID * pformattype  
);
```

### Parameters

*pformattype*  
    GUID describing the format type.

### Return Values

No return value.

### Remarks

The format GUID describes what can be expected to be found in the variable-length format block. For example, if the format type is `FORMAT_VideoInfo`, the format block should contain a VIDEOINFOHEADER structure. The creator of this object is responsible for making them consistent.

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CMediaType::SetSampleSize

### CMediaType Class

Sets the maximum sample size for the data stream.

```
void SetSampleSize(  
    ULONG sz  
);
```

### Parameters

*sz*  
    Size of the sample.

### Return Values

No return value.

### Remarks

If the sample size passed is zero, the object is set so that the data stream will send variable-length samples (the `CMediaType::GetSampleSize` member function will return zero). Otherwise, it will set the maximum size of the sample to the size specified in the `sz` parameter.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::SetSubtype

[CMediaType Class](#)

Sets the subtype for the object.

```
void SetSubtype(  
    const GUID * psubtype  
);
```

### Parameters

*psubtype*  
[GUID](#) defining the subtype for the object.

### Return Values

No return value.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::SetTemporalCompression

[CMediaType Class](#)

Marks the media type so that the data stream it describes might or might not contain temporal compression (according to the input Boolean flag).



```
void SetTemporalCompression(  
  BOOL bCompressed  
  );
```

### Parameters

*bCompressed*

TRUE to indicate that the stream will contain temporal compression; otherwise, FALSE.

### Return Values

No return value.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::SetType

[CMediaType Class](#)

Sets the major type for the object.

```
void SetType(  
  const GUID * pType  
  );
```

### Parameters

*pType*

[GUID](#) defining the major type for the object.

### Return Values

No return value.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::SetVariableSize

### CMediaType Class

Sets the media type to indicate that the data stream will send variable-length samples.

**void SetVariableSize( );**

#### **Return Values**

No return value.

#### **Remarks**

Subsequent calls to CMediaType::GetSampleSize will return zero.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::Subtype

### CMediaType Class

Retrieves the subtype.

**const GUID \*Type( ) const;**

#### **Return Values**

Returns a pointer to the subtype.

#### **Remarks**

The subtype GUID gives finer detail within the major type of data represented by this media type.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMediaType::Type

### CMediaType Class

Retrieves the major type.

```
const GUID *Type( ) const;
```

### Return Values

Returns a pointer to the major type.

### Remarks

The major type GUID describes the class of data represented by this media type.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::operator =

### CMediaType Class

The CMediaType variation of this operator is the copy constructor for a **CMediaType** object.

The AM\_MEDIA\_TYPE variation of this operator is the copy constructor for an **AM\_MEDIA\_TYPE** object.

```
CMediaType& operator=(  
    const CMediaType& rt  
);  
CMediaType& operator=(  
    const AM_MEDIA_TYPE& mrt  
);
```

### Parameters

*rt* Object to copy during the assignment operation.  
*mrt* Object to copy during the assignment operation.

### Return Values

Returns a reference to this object after the operation.

### Remarks

Because the CMediaType class inherits publicly from AM\_MEDIA\_TYPE, the compiler could generate the copy constructor for the **AM\_MEDIA\_TYPE** object itself. However, this could

introduce some memory conflicts and leaks in the process because the structure contains a dynamically allocated block (which the **AM\_MEDIA\_TYPE** `pbFormat` member points to), which the compiler's copy constructor will not copy correctly.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::operator ==

### [CMediaType Class](#)

Tests for equality between [CMediaType](#) objects.

```
inline BOOL operator==(const CMediaType& rt) const;
```

### Parameters

*rt*  
[CMediaType](#) object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the [CMediaType](#) object tested is equal to this object; otherwise, returns FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMediaType::operator !=

### [CMediaType Class](#)

Tests for inequality between [CMediaType](#) objects.

```
BOOL operator!=(  
    const CMediaType& rt  
    ) const;
```

**Parameters***rt*

CMediaType object corresponding to the right side of the operator.

**Return Values**

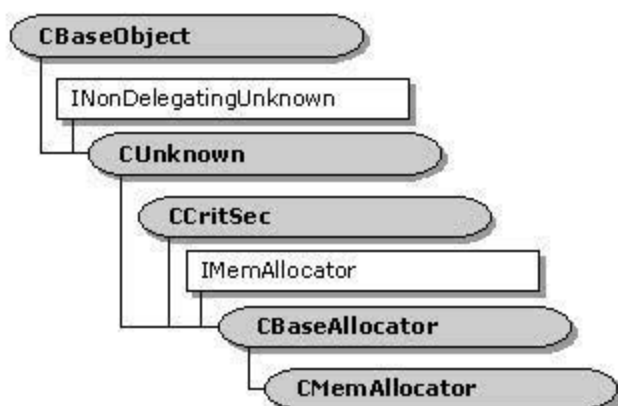
Returns TRUE if the CMediaType object tested is not equal to this object; otherwise, returns FALSE.

**Remarks**

This object is on the left side of the operator.

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

## CMemAllocator Class



This class provides support for [IMemAllocator](#) by using the **new** operator to allocate memory. Derived from [CBaseAllocator](#), it overrides the [CBaseAllocator::Alloc](#) member function to allocate a single block of memory large enough to hold all the requested data areas, and then allocates (using the **new** operator) a [CMediaSample](#) object for each requested buffer pointing into the data area.

The [CBaseInputPin](#) and [CBaseOutputPin](#) classes instantiate [CMemAllocator](#) objects as the default allocator if no other suitable allocator is provided.

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
<a href="#">Alloc</a>	Allocates memory for a media sample (overrides <a href="#">CBaseAllocator::Alloc</a> ).
<a href="#">CMemAllocator</a>	Constructs a <a href="#">CMemAllocator</a> object.
<a href="#">ReallyFree</a>	Frees memory when called from the destructor (or from <a href="#">Alloc</a> when reallocating for new size or count).

### Overridable Member Functions

Name	Description
<a href="#">Free</a>	Indicates an overridden <a href="#">CBaseAllocator::Free</a> member function, called when a decommit operation is complete to free memory.

### Implemented IMemAllocator Methods

Name	Description
<a href="#">CreateInstance</a>	Creates new instances of <a href="#">CMemAllocator</a> in the factory template.
<a href="#">SetProperties</a>	Sets the number of media samples and the size of each.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMemAllocator::Alloc

[CMemAllocator Class](#)

Allocates a media sample object.

**HRESULT Alloc(void);**

### Return Values

Returns an [HRESULT](#) value.

### Remarks

This member function instantiates [CMediaSample](#) objects, adds them to the [m\\_IFree](#) data members, and updates the [m\\_IAllocated](#) count. This member function is called from [IMemAllocator::Commit](#) when becoming active.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMemAllocator::CMemAllocator

[CMemAllocator Class](#)

Constructs a [CMemAllocator](#) object.

```
CMemAllocator(  
    TCHAR * pName,  
    LPUNKNOWN lpUnk,  
    HRESULT * phr  
);
```

### Parameters

*pName*

Name of the allocator object.

*lpUnk*

Pointer to LPUNKNOWN.

*phr*

Pointer to the general COM return value. Note that this value is changed only if this function fails.

### Return Values

No return value.

### Remarks

This constructor is passed to [CBaseAllocator::CBaseAllocator](#), which initializes the data members.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMemAllocator::CreateInstance

[CMemAllocator Class](#)

Creates new instances of [CMemAllocator](#) in the factory template.

```
static CUnknown *CreateInstance(  
    LPUNKNOWN pUnk,  
    HRESULT *phr  
);
```

### Parameters

*pUnk*

Pointer to the [IUnknown](#) interface.

*phr*

Pointer to the [HRESULT](#) value into which to place resulting information.

### Return Values

Returns the *pUnkRet* parameter, which is a [CUnknown](#) class object.

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



[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMemAllocator::Free

[CMemAllocator Class](#)

Frees memory for a media sample object.

**HRESULT Free(void);**

### Return Values

No return value.

### Remarks

This member function overrides the pure virtual [CBaseAllocator::Free](#) member function called when a decommit operation has completed. Memory is actually freed in [ReallyFree](#), which is called from the destructor, so this function is not used in this class.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMemAllocator::ReallyFree

[CMemAllocator Class](#)

Releases all media samples in the free list.

**void ReallyFree (void);**

### Return Values

No return value.

### Remarks

The [CMemAllocator](#) class holds memory until the object is actually deleted. This member function can be overridden to handle freeing media samples when a decommit occurs.

This member function is protected.

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## CMemAllocator::SetProperties

### CMemAllocator Class

Determines the size, number, and alignment of blocks.

```
HRESULT SetProperties(  
    ALLOCATOR_PROPERTIES * pRequest,  
    ALLOCATOR_PROPERTIES * pActual  
);
```

### Parameters

*pRequest*

Requested allocator properties.

*pActual*

Allocator properties actually set.

### Return Values

Returns an [HRESULT](#) value.

### Remarks

The *pRequest* parameter is filled in by the caller with the requested values for the count, number, and alignment as specified by the [ALLOCATOR\\_PROPERTIES](#) structure. The *pActual* parameter is filled in by the allocator with the closest values it can provide for the request. This member function cannot be called unless the allocator has been decommitted using the [IMemAllocator::Decommit](#) method.

This member function replaces **SetCountAndSize** in previous releases.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## CMsg Class

### CMsg

The [CMsgThread](#) class provides support for a worker thread to which requests can be posted asynchronously instead of sent directly. The [CAMThread](#) class provides a worker thread to which single requests can be sent. Only one client can make a request at a time, and the client blocks until the worker thread has completed the request. By contrast, the **CMsgThread** class provides a worker thread to which any number of requests can be posted. The requests (in the form of a **CMsg** object) are queued and executed in order, asynchronously. No reply or return value is received.

#### Data Members

Name	Description
------	-------------

<b>dwFlags</b>	Flag parameter to the request code.
----------------	-------------------------------------

<b>lpParam</b>	Data required by the worker thread as parameter or return values. This data should not be stack-based, as it will be referenced some time after completing the queuing operation.
----------------	---

<b>pEvent</b>	Event object that a worker thread can signal to indicate the completion of the operation.
---------------	---

<b>uMsg</b>	Request code that is defined by the client of the thread class and understood by the overridden worker thread function.
-------------	---

#### Member Functions

Name	Description
------	-------------

<a href="#">CMsg</a>	Constructs a <a href="#">CMsg</a> object.
----------------------	---

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsg::CMsg

### [CMsg Class](#)

Constructs a [CMsg](#) object.

```
CMsg(  
  UINT u,  
  DWORD dw,  
  LPVOID lp,  
  CAMEvent *pEvent  
);
```

### Parameters

*u*

Request code, defined by the client of the thread class and understood by the overridden worker thread function.

*dw*

Flag parameter to the request code.

*lp*

Data required by the worker thread as parameter or return values. This data should not be stack-based, as it will be referenced some time after completing the queuing operation.

*pEvent*

Event object that a worker thread can signal to indicate the completion of the operation.

### Return Values

No return value.

### Remarks

This member function contains a request for a [CMsgThread](#) worker thread to act on. All the parameters are passed to the worker thread function as parameters when this message gets processed. The meanings of the parameters are defined by the client function that calls the worker thread and the derived class that supplies the worker thread's execution function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

## CMsgThread Class

### CMsgThread

This class is a worker thread class that queues requests to the queuing thread for completion asynchronously. To use this class, derive your class from it and override the [CMsgThread::ThreadMessageProc](#) member function. The [ThreadMessageProc](#) member function carries out each request. Your client functions and the **ThreadMessageProc** member function must share a common definition of the parameters in the [CMsg](#) object.

A negotiated mechanism tells the worker thread to exit. Typically, this will be one value of the [CMsg](#) class's [uMsg](#) message code.

It is a good idea to send this message from the destructor of your derived class, and call the [CMsgThread::WaitForThreadExit](#) member function before completing the destruction of the derived class.

#### Protected Data Members

Name	Description
<a href="#">m_hSem</a>	Indicates a handle used for signaling.
<a href="#">m_Lock</a>	Protects access to lists.
<a href="#">m_lWaiting</a>	Indicates waiting for a free thread.
<a href="#">m_ThreadPool</a>	Overrides the <a href="#">CMsgThread::GetThreadMsg</a> member function and blocks on things other than this queue.

#### Member Functions

Name	Description
<a href="#">CMsgThread</a>	Constructs a <a href="#">CMsgThread</a> object.
<a href="#">CreateThread</a>	Creates a thread.
<a href="#">GetThreadHandle</a>	Returns the thread handle.
<a href="#">GetThreadID</a>	Returns the identifier of the thread.
<a href="#">GetThreadPriority</a>	Retrieves the current thread priority.
<a href="#">PutThreadMsg</a>	Queues a request for execution by the worker thread.
<a href="#">ResumeThread</a>	Continues the operation of the worker thread.
<a href="#">SetThreadPriority</a>	Sets the priority of the thread to a new value.
<a href="#">SuspendThread</a>	Suspends the operation of a running thread.
<a href="#">WaitForThreadExit</a>	Blocks until the thread has exited after a call to the <a href="#">CMsgThread::SuspendThread</a> member function.

#### Overridable Member Functions

Name	Description
<a href="#">GetThreadMsg</a>	Retrieves a queued <a href="#">CMsg</a> object containing a request.
<a href="#">OnThreadInit</a>	Provides initialization on a thread.
<a href="#">ThreadMessageProc</a>	Processes requests. This is a pure virtual member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::CMsgThread

[CMsgThread Class](#)

Constructs a [CMsgThread](#) object.

**CMsgThread( );**

**Return Values**

No return value.

**Remarks**

Constructing a message thread object does not automatically create the thread. You must call the [CMsgThread::CreateThread](#) member function to create the thread.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::CreateThread

[CMsgThread Class](#)

Creates a thread.

**BOOL CreateThread( );**

**Return Values**

Returns one of the following values.

**Value Meaning**

TRUE Thread was successfully created.

FALSE Thread was not successfully created.

**Remarks**

The thread will loop, blocking until a request is queued (through the [CMsgThread::PutThreadMsg](#) member function) and then calling the [CMsgThread::ThreadMessageProc](#) member function with each message.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::GetThreadHandle

### [CMsgThread Class](#)

Retrieves the handle to the thread in the [CMsgThread](#) object.

**HANDLE GetThreadHandle( );**

**Return Values**

Returns the thread handle.

**Remarks**

The thread handle can be passed to Microsoft® Win32® application programming interface (API) functions, such as [WaitForMultipleObjects](#). The thread handle is signaled when the thread has exited.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::GetThreadID

### [CMsgThread Class](#)

Retrieves the thread's identifier.

## DWORD GetThreadID( );

### Return Values

Returns the *m\_ThreadId* private data member.

### Remarks

This function returns the Microsoft Win32 identifier for the worker thread. You can call this member function on either the worker thread or a client thread.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::GetThreadMsg

### CMsgThread Class

Retrieves a queued CMsg object containing a request.

```
void virtual GetThreadMsg(  
    CMsg *msg  
);
```

### Parameters

*msg*  
Pointer to an allocated CMsg object.

### Remarks

This member function is called from the worker thread's private ThreadProc function to retrieve the next member function. The *msg* parameter should point to an allocated CMsg object that will be filled with the parameters to the next request in the queue. If there are no queued requests, this member function blocks until the next request is queued (by a call to the CMsgThread::PutThreadMsg member function).

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::GetThreadPriority



### CMsgThread Class

Uses the Microsoft Win32 **GetThreadPriority** function to retrieve the priority of the current worker thread.

**int GetThreadPriority( );**

#### **Return Values**

Returns the thread priority as an integer.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## **CMsgThread::OnThreadInit**

### CMsgThread Class

Provides initialization on a thread.

**virtual void OnThreadInit( );**

#### **Return Values**

No return value.

#### **Remarks**

Override this function if you want to do your own specific initialization on thread startup.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## **CMsgThread::PutThreadMsg**

### CMsgThread Class

Queues a request for execution by the worker thread.

```
void PutThreadMsg(  
    UINT uMsg,  
    DWORD dwMsgFlags,  
    LPVOID lpMsgParam,  
    CAMEvent *pEvent = NULL  
);
```

### Parameters

*uMsg*

Request code.

*dwMsgFlags*

Optional flags parameter.

*lpMsgParam*

Optional pointer to a data block containing additional parameters or return values. Must be statically or heap-allocated and not automatic.

*pEvent*

Optional pointer to an event object to be signaled upon completion.

### Return Values

No return value.

### Remarks

This member function queues a request for execution by the worker thread. The parameters of this member function will be queued (in a CMsg object) and passed to the CMsgThread::ThreadMessageProc member function of the worker thread. This member function returns immediately after queuing the request and does not wait for the thread to fulfill the request. The **CMsgThread::ThreadMessageProc** member function of the derived class defines the four parameters.

This member function uses a multithread safe list, so multiple calls to this member function from different threads can be made safely.

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

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## CMsgThread::ResumeThread

### CMsgThread Class

Uses the Microsoft Win32 **ResumeThread** function to continue the operation of the worker thread after a previous call to the CMsgThread::SuspendThread member function.

**DWORD ResumeThread( );**

### Return Values

If the member function succeeds, the return value is the previous suspend count of the thread. If the member function fails, the return value is 0xFFFFFFFF. To obtain extended error information, call the Microsoft Win32 [GetLastError](#) function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::SetThreadPriority

[CMsgThread Class](#)

Uses the Microsoft Win32 **SetThreadPriority** function to set the priority of the thread to a new value.

**BOOL SetThreadPriority(**  
**int nPriority**  
**);**

### Parameters

*nPriority*  
Priority of the thread.

### Return Values

Returns one of the following values.

#### Value Meaning

TRUE Priority was successfully set.  
FALSE Priority was not set.

### Remarks

The client and the worker thread can call this member function.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::SuspendThread

### CMsgThread Class

Uses the Microsoft Win32 **SuspendThread** function to suspend the operation of a running thread.

**DWORD SuspendThread( );**

### Return Values

If the member function succeeds, the return value is the previous suspend count of the thread. If the member function fails, the return value is 0xFFFFFFFF. To obtain extended error information, call the Microsoft Win32 GetLastError function.

### Remarks

The client thread calls this member function to suspend the operation of the worker thread. The worker thread remains suspended and will not execute until an additional call to the CMsgThread::ResumeThread member function is made.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## CMsgThread::ThreadMessageProc

### CMsgThread Class

Processes requests. This is a pure virtual member function.

```
virtual HRESULT ThreadMessageProc(  
    UINT uMsg,  
    DWORD dwFlags,  
    LPVOID lpParam,  
    CAMEvent *pEvent  
);
```

### Parameters

*uMsg*  
Request code.

*dwFlags*

Optional flag parameter to request.

*lpParam*

Optional pointer to extra data or a return data block.

*pEvent*

Optional pointer to an event object.

## Return Values

Any nonzero return causes the thread to exit. Returns zero unless an exit request has been processed recently.

## Remarks

This pure virtual function must be overridden in your derived class. It will be called once for each request queued by a call to the `CMsgThread::PutThreadMsg` member function.

The member function defines the four parameters. Typically, use the *uMsg* parameter to indicate the request, and the other three parameters will be optional additional parameters. The calling application can supply a pointer to a `CAMEvent` object in the *pEvent* parameter if your application requires it. You must set this event after processing the event by using an expression such as:

```
pEvent->SetEvent
```

One request code must be set aside to tell the worker thread to exit. Upon receiving this request, return 1 from this member function. Return 0 if you do not want the worker thread to exit.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

# CMsgThread::WaitForThreadExit

CMsgThread Class

Blocks until the thread exits.

```
BOOL WaitForThreadExit(
    LPDWORD lpdwExitCode
);
```

## Parameters

*lpdwExitCode*

Exit code returned by the thread.

**Return Values**

Returns either TRUE or FALSE, the meaning of which is determined by the class supplying the overridden CMsgThread::ThreadMessageProc member function and the calling member function.

**Remarks**

Ensure that the worker thread has exited completely before completing the destruction of your derived class; otherwise, the thread might still execute after your dynamic-link library (DLL) has been unloaded from the address space of the process. Even if the only instruction left to exit is a single-return instruction, this would cause an exception. The only reliable way to ensure that the thread has exited is to signal the thread to exit (using a privately negotiated CMsg object sent to the CMsgThread::PutThreadMsg member function) and then call this member function. You should do this in the destructor for your derived class.

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

## COARefTime Class



This class converts between the Automation-compatible REFTIME type and the REFERENCE\_TIME type used within and between filters.

Time parameters within the control interfaces are represented as double values, containing a fractional number of seconds. Interfaces supported between and within filters use a 64-bit LONGLONG type containing the time in 100-nanosecond units. Filters use this class to convert between the two formats. It is derived from CRefTime and thus is a **LONGLONG**, but it can be constructed from and assigned **double** values.

### Member Functions

Name	Description
<u>COARefTime</u>	Constructs a <u>COARefTime</u> object.

### Operators

Name	Description
<u>double</u>	Returns the reference time as a <u>double</u> value.
<u>Operator =</u>	Copy constructor for the <u>COARefTime</u> class.
<u>Operator ==</u>	Tests for equality between <u>COARefTime</u> objects.
<u>Operator !=</u>	Tests for inequality between <u>COARefTime</u> objects.
<u>Operator &lt;</u>	Tests if one <u>COARefTime</u> object is less than another <b>COARefTime</b> object.
<u>Operator &gt;</u>	Tests if one <u>COARefTime</u> object is greater than another <b>COARefTime</b> object.
<u>Operator &lt;=</u>	Tests if one <u>COARefTime</u> object is less than or equal to another <b>COARefTime</b> object.
<u>Operator &gt;=</u>	Tests if one <u>COARefTime</u> object is greater than or equal to another <b>COARefTime</b> object.
<u>Operator +</u>	Adds two <u>COARefTime</u> objects.
<u>Operator -</u>	Subtracts one <u>COARefTime</u> object from another <b>COARefTime</b> object.
<u>Operator +=</u>	Adds two <u>COARefTime</u> objects and makes this object equal to the result.
<u>Operator -=</u>	Subtracts one <u>COARefTime</u> object from another <b>COARefTime</b> object and makes this object equal to the result.
<u>Operator *</u>	Multiplies two <u>COARefTime</u> objects.
<u>Operator /</u>	Divides one <u>COARefTime</u> object by another <b>COARefTime</b> object.

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

---

## COARefTime::COARefTime

[COARefTime Class](#)

Constructs a [COARefTime](#) object.

```
COARefTime(  
    CRefTime t  
);  
COARefTime(  
    REFERENCE_TIME t  
);  
COARefTime(  
    double d  
);
```

### Parameters

*t*  
A [CRefTime](#) value or REFERENCE\_TIME value passed through to the class. Units are 100 nanoseconds.

*d*  
A [double](#) value that constructs the [COARefTime](#) class. Units in this case are (fractional) seconds.

### Return Values

No return value.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

---

## COARefTime::double

[COARefTime Class](#)

Retrieves the reference time as a [double](#) value, converted from 100-nanosecond units to



seconds.

```
operator double ();
```

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator =

[COARefTime Class](#)

Copy constructor for a [COARefTime](#) object.

```
COARefTime& operator=(  
  const double& rd  
  );
```

### Parameters

*rd*

A [double](#) value that constructs a [COARefTime](#) object.

### Return Values

No return value.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator ==

[COARefTime Class](#)

Tests for equality between [COARefTime](#) objects.

```
BOOL operator==(  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
COARefTime object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the COARefTime object tested is equal to this object and FALSE otherwise.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator !=

COARefTime Class

Tests for inequality between COARefTime objects.

```
BOOL operator!=(  
    const COARefTime& rt  
);
```

### Parameters

*rt*  
COARefTime object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the COARefTime object tested is not equal to this object; otherwise, returns FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator <

[COARefTime Class](#)

Tests if one [COARefTime](#) object is less than another **COARefTime** object.

```
BOOL operator < (  
    const COARefTime& rt  
    );
```

### Parameters

*rt*  
    [COARefTime](#) object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the [COARefTime](#) object tested is less than this object; otherwise, returns FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## COARefTime::Operator >

[COARefTime Class](#)

Tests if one [COARefTime](#) object is greater than another **COARefTime** object.

```
BOOL operator > (  
    const COARefTime& rt  
    );
```

### Parameters

*rt*  
    [COARefTime](#) object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the [COARefTime](#) object tested is greater than this object; otherwise, returns

FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator >=

[COARefTime Class](#)

Tests if one [COARefTime](#) object is greater than or equal to another **COARefTime** object.

```
BOOL operator >= (  
    const COARefTime& rt  
);
```

### Parameters

*rt*  
    [COARefTime](#) object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the [COARefTime](#) object tested is greater than or equal to this object; otherwise, returns FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator <=

[COARefTime Class](#)

Tests if one [COARefTime](#) object is less than or equal to another **COARefTime** object.

```
BOOL operator <= (  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
COARefTime object corresponding to the right side of the operator.

### Return Values

Returns TRUE if the COARefTime object tested is less than or equal to this object; otherwise, returns FALSE.

### Remarks

This object is on the left side of the operator.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator +

### COARefTime Class

Adds two COARefTime objects.

```
COARefTime operator+(  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
COARefTime object to be added.

### Return Values

Returns the result of the addition.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator -

[COARefTime Class](#)

Subtracts one [COARefTime](#) object from another **COARefTime** object.

```
COARefTime operator-(  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
 [COARefTime](#) object to be subtracted.

### Return Values

Returns the result of the subtraction.

### Remarks

This object is the object subtracted from.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator +=

[COARefTime Class](#)

Adds two [COARefTime](#) objects and makes this object equal to the result.

```
COARefTime& operator+=(  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
 [COARefTime](#) object to be added.

### Return Values

Returns the result.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator -=

[COARefTime Class](#)

Subtracts one [COARefTime](#) object from another **COARefTime** object and makes this object equal to the result.

```
COARefTime& operator-=(  
  const COARefTime& rt  
  );
```

### Parameters

*rt*  
 [COARefTime](#) object to be subtracted.

### Return Values

Returns the result.

### Remarks

This object is the object subtracted from.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COARefTime::Operator \*

[COARefTime Class](#)

Multiplies the [COARefTime](#) object by a value.

```
COARefTime operator*(  
  LONG /
```

```
);
```

### Parameters

```
/
```

Value to multiply by.

### Return Values

Returns the result.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)

[Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next](#)

---

## COARefTime::Operator /

### COARefTime Class

Divides one COARefTime object by a value.

```
COARefTime operator /(  
    LONG /  
);
```

### Parameters

```
/
```

Value to divide by.

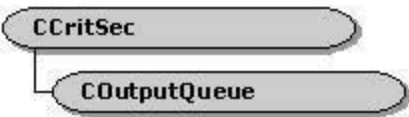
### Return Values

Returns the result.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use.](#)



## COutputQueue Class



```

classDiagram
    class COutputQueue
    class CCritSec
    COutputQueue --|> CCritSec
  
```

Output pins use the **COutputQueue** to send samples to another filter by using the local memory-based transport (that is, to input pins that support the [IMemInputPin](#) interface). **COutputQueue** uses [IMemInputPin::ReceiveCanBlock](#) to determine if the connected input pin has a blocking implementation of [IMemInputPin::Receive](#). If so, all samples are queued in **COutputQueue** and a thread is created to pass samples from the queue to the connected input pin. If the input pin's [IMemInputPin::Receive](#) method does not block, samples are passed directly to [IMemInputPin::Receive](#). **COutputQueue** can also batch samples to reduce the number of calls to the downstream pin.

**COutputQueue** is useful when the filter has other work to do while samples that it has already completed are being processed downstream. This occurs, for example, in a filter that can read more data off disk while data is being processed, or when it has more than one output pin and does not want to starve an output pin because [IMemInputPin::Receive](#) has no optional batching of samples.

To use this class, create one **COutputQueue** object for every output pin for which it will be used. This can either be created when the pin is created and deleted when the pin is disconnected, or it can be created when the pin goes active and deleted when the pin goes inactive.

The samples sent to this object by calling its [COutputQueue::Receive](#) or [COutputQueue::ReceiveMultiple](#) member function should have references added by means of [IUnknown::AddRef](#) (as they usually are if they were obtained directly from an allocator). This object then calls [IUnknown::Release](#) on all samples it receives, whether they were processed successfully or not. Note that [Release](#) is not called for special (control) samples.

Some control information, such as end of stream, needs to be queued with the data and processed once all the data has been delivered. This information is queued in the form of special control packets. **COutputQueue** implements a sticky [HRESULT](#) so it will not send any more data after it gets a return code that is not [S\\_OK](#) from the downstream [ReceiveMultiple](#) call. (A sticky state setting is one that persists even after execution of operations that would normally reset the setting.) This sticky state is reset by the [EndFlush](#) and [EOS](#) calls. However, if the sticky [HRESULT](#) is not [S\\_OK](#), [EOS](#) itself is not sent downstream; the [HRESULT](#) is just reset. Because of this, if this object is not deleted when the pin goes inactive, [BeginFlush](#) and [EndFlush](#) should be called at that time to free the state.

In many ways this object acts as a proxy for the connected input pin, supporting a similar set of methods for stream control.

### Protected Data Members

Name	Description
<b>m_bBatchExact</b>	TRUE if commands are batched; FALSE if commands are sent singly.
<b>m_bFlushed</b>	Flag to signify if samples have been flushed.
<b>m_bFlushing</b>	Flag for flushing state.
<b>m_bSendAnyway</b>	Flag to override batch processing.
<b>m_bTerminate</b>	Termination flag.
<b>m_evFlushComplete</b>	Event signaling that flushing has finished.
<b>m_hSem</b>	Handle used for signaling.
<b>m_hr</b>	<u>HRESULT</u> structure for return values; used to implement a sticky return value (one that persists even after operations that would normally change the value).
<b>m_hThread</b>	Worker thread handle.
<b>m_IBatchSize</b>	Work in batches of this batch size. Ignored if <u>m_bBatchExact</u> is not TRUE.
<b>m_List</b>	Pointer to a <b>CSampleList</b> object. The class <b>CSampleList</b> is a generic list ( <u>CGenericList</u> ) of objects of <u>IMediaSample</u> type. It is defined as follows:  <pre>typedef CGenericList&lt;IMediaSample&gt; CSampleList;</pre>
<b>m_IWaiting</b>	Variable set to nonzero value when waiting for a free element.
<b>m_nBatched</b>	Number of samples currently batched awaiting processing.
<b>m_pInputPin</b>	Pointer to the connected input pin.
<b>m_pPin</b>	Pointer to the output pin.
<b>m_ppSamples</b>	Pointer to an array of batched samples.

### Member Functions

Name	Description
<u>BeginFlush</u>	Causes all unsent samples to be discarded and sets flushing state.
<u>COutputQueue</u>	Constructs a <u>COutputQueue</u> object.
<u>EndFlush</u>	Finalizes flush of batched or queued samples and resets flushing state.
<u>EOS</u>	Queues an end-of-stream call to the connected input pin after all batched and queued samples have been passed to the input pin.
<u>FreeSamples</u>	Removes and releases batched and queued samples.
<u>InitialThreadProc</u>	Executed by the thread on thread creation.
<u>IsIdle</u>	Determines if the output queue is idle.
<u>IsQueued</u>	Determines if samples are being queued or being sent directly.
<u>IsSpecialSample</u>	Determines if the sample is a control sample.
<u>NotifyThread</u>	Notifies the thread that there is something to do.
<u>NewSegment</u>	Queues an <u>IPin::NewSegment</u> call to the connected input pin after all queued samples have been passed to the input pin.
<u>QueueSample</u>	Queues the prepared sample.
<u>Receive</u>	Passes in a single sample to send to the input pin.
<u>ReceiveMultiple</u>	Passes a set of samples to send to the input pin.
<u>Reset</u>	Resets the deferred return code <u>m_hr</u> to allow the output queue to be ready for more data.
<u>SendAnyway</u>	Frees any batches samples to be sent to the input pin.

[ThreadProc](#) Implements the thread that sends samples downstream.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::BeginFlush

[COutputQueue Class](#)

Causes all unsent samples to be discarded and sets the flushing state.

**void BeginFlush( );**

### Return Values

No return value.

### Remarks

This member function calls **BeginFlush** on the connected input pin.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::COutputQueue

[COutputQueue Class](#)

Constructs a [COutputQueue](#) object.

**COutputQueue(**  
**IPin \*pInputPin,**  
**HRESULT \*pHr,**  
**BOOL bAuto = TRUE,**  
**BOOL bQueue = TRUE,**  
**LONG lBatchSize,**  
**BOOL bBatchExact,**  
**LONG lListSize,**  
**DWORD dwPriority**

);

## Parameters

*pInputPin*

Connected pin to which to send data.

*phr*

HRESULT return code.

*bAuto*

If TRUE, the queuing mode is determined by asking the connected input pin if the pin can block (by calling IMemInputPin::ReceiveCanBlock). If FALSE, queued or direct mode is set by the *bQueue* parameter.

*bQueue*

Determines if samples are queued for delivery by a worker thread or are being sent directly. Ignored if *bAuto* is TRUE.

*lBatchSize*

Size of the batch (1 for no batching).

*bBatchExact*

Batch exactly to *lBatchSize* (but use SendAnyway to override batching).

*lListSize*

Likely number in the list.

*dwPriority*

Priority given to the created thread.

## Return Values

No return value.

## Remarks

The *phr* parameter should be updated only to report errors. Usually *bAuto* will be TRUE. In that case, the constructor calls IMemInputPin::ReceiveCanBlock on the downstream pin to determine whether to create a thread, and so to send samples asynchronously. If *bAuto* is FALSE, a thread is created if, and only if, *bQueue* is TRUE.

If the batch size is not 1, data is not sent until *lBatchSize* samples have been received by the object. The exceptions are that, if fewer than *lBatchSize* samples are passed to COutputQueue::Receive or COutputQueue::ReceiveMultiple in this object and *bBatchExact* is FALSE, the samples will be sent anyway.

If *bBatchExact* is TRUE, the COutputQueue::SendAnyway member function will cause the samples to be sent to the thread (if the thread is created).

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

## COutputQueue::EndFlush

### COutputQueue Class

Finalizes flush of batched or queued samples and resets the flushing state.

**void EndFlush( );**

### **Return Values**

No return value.

### **Remarks**

The downstream pin is guaranteed not to block at this stage.

[© 1997 Microsoft Corporation. All rights reserved. Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::EOS

### COutputQueue Class

Queues an end-of-stream call to the connected input pin after all batched and queued samples have been passed to the input pin.

**void EOS( );**

### **Return Values**

No return value.

### **Remarks**

The end-of-stream call is queued as a special control packet when in a queued mode. This member function does not actually send an end-of-stream packet if the `m_hr HRESULT` value is not `S_OK` when it is time to make the call.

[© 1997 Microsoft Corporation. All rights reserved. Terms of Use.](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::FreeSamples

### COutputQueue Class

Removes and releases batched and queued samples.

```
void FreeSamples( );
```

### Return Values

No return value.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::InitialThreadProc

### COutputQueue Class

Implements the static member function that the thread executes on thread creation.

```
static DWORD WINAPI InitialThreadProc(  
    LPVOID pv  
);
```

### Parameters

*pv*  
The **this** pointer for the COutputQueue object.

### Return Values

The derived class defines the meaning of the return value.

### Remarks

On thread creation, the worker thread executes this static function with a pointer to the COutputQueue object as the parameter. This function simply calls the COutputQueue::ThreadProc member function of that object (that is, the function pointed to by *pv*).

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::IsIdle

### COutputQueue Class

Determines if the output pin is idle.

**BOOL IsIdle( );**

#### **Return Values**

Returns TRUE if no threads are in the queue, all data has been sent, and nothing is in the batch. Returns FALSE otherwise.

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

[◀ Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next ▶](#)

---

## COutputQueue::IsQueued

### COutputQueue Class

Determines if the COutputQueue object is in queued or direct mode.

**BOOL IsQueued( );**

#### **Return Values**

Returns one of the following values.

##### **Value Meaning**

TRUE In queued mode. Samples are delivered asynchronously by a worker thread.

FALSE In direct mode. Receive calls are passed synchronously to the input pin.

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

[◀ Previous](#)[Home](#)[Topic Contents](#)[Index](#)[Next ▶](#)

---

## COutputQueue::IsSpecialSample

COutputQueue Class

Determines if a sample is one of the special control samples (containing no data).

```
BOOL IsSpecialSample(  
  IMediaSample *pSample  
  );
```

**Parameters**

*pSample*

Pointer to the sample to be passed to the connected input pin.

**Return Values**

Returns one of the following values.

**Value Meaning**

TRUE *pSample* is a special control sample.

FALSE *pSample* is an IMediaSample interface.

**Remarks**

Special control samples are queued in line with the data by methods (such as COutputQueue::EOS) that require processing once all queued data has been delivered. The COutputQueue::ThreadProc member function detects these special samples on the queue by using **IsSpecialSample** and processes them appropriately.

A special sample is one of following types and contains no media data.

EOS\_PACKET

NEW\_SEGMENT

RESET\_PACKET

SEND\_PACKET

Special control samples are relevant only if you plan to change or extend the default base class implementation of COutputQueue in a derived class. Normal use of the **COutputQueue** class does not require the use of control samples.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::NewSegment



COutputQueue Class

Queues an [IPin::NewSegment](#) call to the connected input pin after all queued samples have been passed to the input pin.

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

**Parameters**

*tStart*  
[in] Start time of the segment.

*tStop*  
[in] Stop time of the segment.

*dRate*  
[in] Rate of the segment.

**Return Values**

Returns an [HRESULT](#) value.

**Remarks**

This member function calls the [IPin::NewSegment](#) method on the output pin once all previous data has been delivered. Like [COutputQueue::EOS](#), the **COutputQueue::NewSegment** call and its parameters are queued as a special control sample if the [COutputQueue](#) object is in queued mode, and the [IPin::NewSegment](#) method is called from the worker thread in [COutputQueue::ThreadProc](#).

Special control samples, as implemented by this member function, are only relevant if you plan to change or extend the default base class implementation of [COutputQueue](#) in a derived class. Normal use of the **COutputQueue** class does not require the use of control samples.

This member function allows filters that process buffers containing more than one sample to delineate the rendering of the samples between start and stop time, as indicated by the *tStart* and *tStop* parameters.

**COutputQueue::NewSegment** is intended to be implemented on an input pin. A connected output pin on the upstream filter calls this member function after completing delivery of previous data and before calling [IMemInputPin::Receive](#) with any new data. It indicates that all data arriving after this call is part of a segment delineated by the parameters.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

[◀ Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next ▶](#)

---

## COutputQueue::NotifyThread

### COutputQueue Class

Notifies the thread that there is data on the queue to process.

**void NotifyThread( );**

### Return Values

No return value.

### Remarks

The critical section must be held when this is called.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::QueueSample

### COutputQueue Class

Queues a sample.

**void QueueSample(  
IMediaSample \*pSample  
);**

### Parameters

*pSample*

Pointer to the sample to be passed to the connected input pin.

### Return Values

No return value.

### Remarks

The critical section must be held when this is called.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::Receive

### COutputQueue Class

Passes in a single sample to send to the input pin.

```
HRESULT Receive(  
    IMediaSample *pSample  
);
```

#### Parameters

*pSample*

Pointer to the sample to be passed to the connected input pin.

#### Return Values

Returns an HRESULT value, which can include the following values, or others.

Value	Meaning
-------	---------

S_FALSE	End of stream detected before or while processing sample; any further samples will be discarded and this value returned.
Other	An error occurred before or while processing sample; any further samples will be discarded and this value returned.
S_OK	Queued successfully or passed to the connected input pin if there is no queue.

#### Remarks

If the sticky return code (m\_hr) is not S\_OK, the sample is not sent and the sticky return code is returned. (A sticky return code is one that persists even after operations that would normally change its value.) The samples are all released (by means of Release) after processing, regardless of whether the processing was successful.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::ReceiveMultiple

### COutputQueue Class

Passes a set of samples to send to the input pin.

```
HRESULT ReceiveMultiple (
    IMediaSample **ppSamples,
    long nSamples,
    long *nSamplesProcessed
);
```

### Parameters

*ppSamples*

Pointer to the set of samples to be passed to the connected input pin.

*nSamples*

Number of samples pointed to by *ppSamples*.

*nSamplesProcessed*

Updated to be the number of samples processed.

### Return Values

Returns an [HRESULT](#) value, which can include the following values, or others.

Value	Meaning
Other	An error occurred before or while processing sample; any further samples will be discarded and this value returned.
S_FALSE	End of stream detected before or while processing sample; any further samples will be discarded and this value returned.
S_OK	Queued successfully or passed to the connected input pin if there is no queue.

### Remarks

If the sticky return code is not S\_OK, the sample is not sent and the sticky return code is returned. (A sticky return code is one that persists even after operations that would normally change its value.) The samples are all released (by means of [Release](#)) after processing, regardless of whether the processing was successful.

© 1997 Microsoft Corporation. All rights reserved. [Terms of Use](#).

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

[◀Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next▶](#)

---

## COutputQueue::Reset

### COutputQueue Class

Resets the deferred return code `m_hr` to ready the output queue for more data.

**void Reset( );**

#### **Return Values**

No return value.

#### **Remarks**

The sticky return code `m_hr` is set to `S_OK` if data is queued; otherwise, this function queues the sample and notifies the thread. (A sticky return code is one that persists even after operations that would normally change its value.)

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::SendAnyway

### COutputQueue Class

If *bBatchExact* was specified on construction, frees batched samples so they can be sent to the input pin.

**void SendAnyway( );**

#### **Return Values**

No return value.

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

[Previous](#) [Home](#) [Topic Contents](#) [Index](#) [Next](#)

---

## COutputQueue::ThreadProc

### COutputQueue Class

Implements the thread that sends samples downstream.

**DWORD ThreadProc( );**

### **Return Values**

Returns zero when DirectShow terminates the thread.

### **Remarks**

This is the main thread procedure for the class, which is called from COutputQueue::InitialThreadProc. It sends a sample or a batch of samples to the connected input pin (depending on the m\_bBatchExact, m\_nBatched, and m\_lBatchSize data members) when conditions are met. Otherwise, it increments the m\_lWaiting data member, while holding the critical section and waits for m\_hSem to be set (not holding the critical section) to continue.

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