Asynchronous Moniker Specification

Last updated: 5/30/96

1Introduction

The Internet requires new approaches to application design, primarily due to its low-speed, high latency network access. In general, applications will want to perform all expensive network operations asynchronously to avoid stalling the user interface. An application triggers an operation and receives a notification on completion (or partial completion). It then proceeds with the next step of the operation or provides additional information needed at that point.

An application will also want to be able to provide users with progress information and the opportunity to cancel an operation at any time.

A common operation in COM/OLE scenarios is that of binding to a moniker, be it for instantiation of an object or to obtain access to a persisted representation. This specification provides for different levels of asynchronous behavior during the bind operation, while providing backwards compatibility for applications either unaware of or not requiring asynchronous behavior.

There are other related technologies that provide asynchronous behavior. Of special significance in this context is Asynchronous Storage. Asynchronous Monikers and Asynchronous Storage will work “hand in hand” to provide a complete asynchronous binding behavior. The moniker triggers the bind operation and sets up the components involved (notifications, objects, etc.). Once the components are connected, the moniker “gets out of the way” and the rest of the bind is executed between the components implementing the storage, the object and eventual coordinating services.

Related documents:

Document	**Filename**
HREF="urlmon.doc	urlmon.doc
HREF="Asynchronous (for review only)	Asynchronous Storage.doc

1.1Overview

The following figures show the components involved in using monikers and asynchronous monikers:

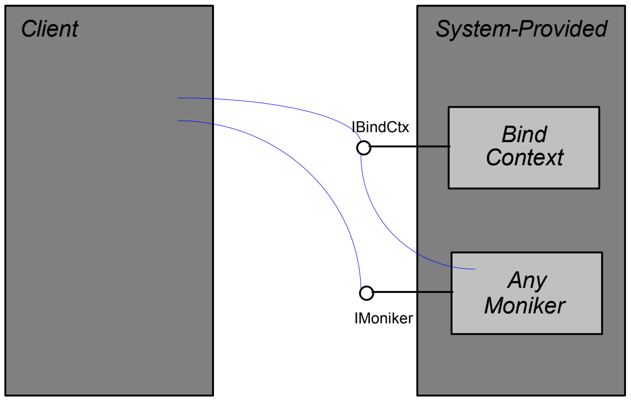

Figure 1. Components Involved in using traditional monikers (small light-gray boxes), who they are implemented by (larger dark-gray boxes), and their references to one-another (dotted lines).

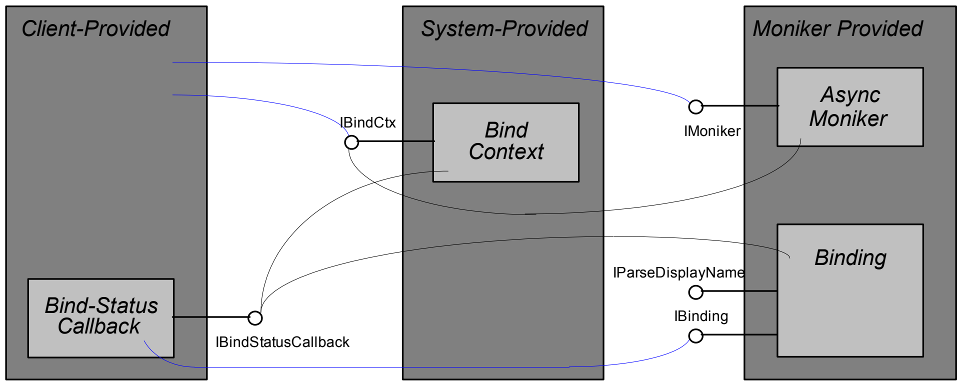

Figure 2. Components Involved in using Asynchronous Monikers (small light-gray boxes), who they are implemented by (larger dark-gray boxes), and their references to one-another (dotted lines).

A client of a standard moniker (Figure 1) typically creates and holds a reference to the *moniker* as well as the *bind-context* to be used during binding (BindToStorage or BindToObject). Standard monikers are typically created through APIs, such as CreateFileMoniker, CreateItemMoniker, or CreatePointerMoniker, or since they are persistable, through OleSaveToStream and OleLoadFromStream.

As shown in the Figure 2, a client of an asynchronous moniker also creates and holds a reference to the moniker and bind-context to be used during binding. The client furthermore implements a *bind-status-callback* object supporting IBindStatusCallback and registers it with the bind-context using RegisterBindStatus-Callback. This callback object receives the IBinding interface associated with the specific bind operation during IBindStatusCallback::OnStartBinding and will tell the asynchronous moniker *how* it wants to bind during IBindStatusCallback::GetBindInfo. The callback object also receives progress notification through IBindStatusCallback::OnProgress, data availability notification through IBindStatusCallback::OnDataAvailable, as well as various other notifications from the moniker about the status of the binding.

1.1.1 Asynchronous vs. Synchronous Binding

Clients may check if a moniker might support asynchronous binding by using the IsAsyncMoniker API. An asynchronous moniker will not return an object or storage from IMoniker::BindToStorage and IMoniker::Bind-ToObject when the BINDF_ASYNCHRONOUS flag is specified by the client in IBindStatusCallback::GetBindInfo. In this case, the moniker returns MK_S_ASYNCHRONOUS and NULL for the resulting object/storage pointer, and the client should wait to receive the requested object or storage during IBindStatusCallback::OnDataAvailable and IBindStatusCallBack::OnObjectAvailable. If, on the other hand, the client does not specify the BINDF_ASYNCHRONOUS flag, the moniker bind operation will proceed synchronously, and the desired object/storage will be returned in IMoniker::BindToObject or IMoniker::BindToStorage.[1]

1.1.2 Asynchronous vs. Synchronous Storage

Asynchronous monikers may also return an asynchronous storage object in the IBindStatusCallback::On-DataAvailable notification resulting from an asynchronous call to IMoniker::BindToStorage. This storage object may allow access to *some* of the data while the binding is still in progress. A client can choose between two modes for the storage: blocking and non-blocking. Blocking mode (the default) is compatible with current implementations of storage objects. If data is not available yet, the call blocks[2] until the data arrives. In non-blocking mode, the storage object returns the error E_PENDING when data is not yet available. An asynchronous aware client notes this error and waits for further notifications (IBindStatusCallback::OnDataAvailable) to retry the operation. A client can choose between a synchronous (blocking) and asynchronous (non-blocking) storage by choosing whether or not to set the BINDF_ASYNCSTORAGE flag in the pgrfBINDF value returned to IBindStatusCallback::GetBindInfo.

[1] Note: this bind operation may be expensive for asynchronous monikers. Asynchronous monikers are usually useful in cases where network latency or bandwidth limitations make it too expensive to bind to objects or data synchronously. However, using these monikers for synchronous binding is useful to support legacy applications and applications that are not able to perform progressive rendering.

[2] In a traditional OLE sense of blocking: some messages are dispatched and IMessageFilter is called.

1.1.3Data-pull model vs. data-push model

A client of an asynchronous moniker can choose between a *data-pull* and *data-push* model for driving an asynchronous IMoniker::BindToStorage operation and receiving asynchronous notifications. In a data-pull model, the client drives the bind operation, and the moniker only provides data to the client when it is read. Specifically, this means that beyond the first call to IBindStatusCallback::OnDataAvailable, the moniker will not provide any data to the client unless the client has consumed *all* of the data that is already available.[3]

On the other hand, in a data-push model, the moniker will drive the asynchronous bind operation and will continuously notify the client whenever new data is available. In such cases, the client of the bind operation may choose whether or not to read the data at any point during the bind operation, but the moniker will continue to drive the bind operation until completion.

1.2Technical Details

```
typedef enum {
        BINDVERB_GET,
        BINDVERB_POST,
        BINDVERB_PUT,
        BINDVERB_CUSTOM
} BINDVERB;

typedef enum {
        BINDINFOF_URLENCODESTGMEDDATA,
        BINDINFOF_URLENCODEDEXTRAINFO,
} BINDINFOF;

typedef struct tagBINDINFO {
        ULONG           cbSize;
        LPWSTR          szExtraInfo;
        STGMEDIUM       stgmedData;          DWORD          grfBindInfoF;
        DWORD           dwBindVerb,
        LPWSTR          szCustomVerb;
        DWORD           cbStgmedData;
} BINDINFO;

typedef enum {
        BINDF_ASYNCHRONOUS,
        BINDF_ASYNCSTORAGE,
        BINDF_PULLDATA,
        BINDF_GETNEWESTVERSION,
        BINDF_NOWRITECACHE
        BINDF_IGNORESECURITYPROBLEM
} BINDF;

typedef enum tagBSCF {
        BSCF_FIRSTDATANOTIFICATION,
        BSCF_LASTDATANOTIFICATION,
        BSCF_INTERMEDIATEDATANOTIFICATION
} BSCF;

typedef enum tagBINDSTATUS {
        BINDSTATUS_FINDINGRESOURCE,
        BINDSTATUS_CONNECTING,
        BINDSTATUS_SENDINGREQUEST,
        BINDSTATUS_REDIRECTING,
        BINDSTATUS_USINGCACHEDCOPY,
        BINDSTATUS_BEGINDOWNLOADDATA,
        BINDSTATUS_DOWNLOADINGDATA,
        BINDSTATUS_ENDDOWNLOADDATA,
        BINDSTATUS_CLASSIDAVAILABLE
} BINDSTATUS;
```

[3] Because data is only downloaded as it is requested, clients that choose the data-pull model must make sure to read this data in a timely manner. In the case of internet-downloads with URL monikers, the bind operation may fail if a client waits to long before requesting more data.

```
interface IBinding : IUnknown {
        HRESULT Abort(void);
        HRESULT Suspend(void);
        HRESULT Resume(void);
        HRESULT SetPriority([in] LONG nPriority);
        HRESULT GetPriority([out] LONG* pnPriority);
        HRESULT GetBindResult( [out] CLSID *pclsidProtocol, [out] DWORD *pdwBindResult,
                                [out] LPWSTR *pszBindResult, [in] DWORD dwReserved );

        };

interface IBindStatusCallback : IUnknown {
        HRESULT GetBindInfo([out] DWORD* pgrfBINDF, [in, out] BINDINFO* pbindinfo);
        HRESULT OnStartBinding([in] DWORD dwReserved, [in] IBinding* pbinding);
        HRESULT GetPriority([out] LONG* pnPriority);
        HRESULT OnProgress( [in] ULONG ulProgress, [in] ULONG ulProgressMax, [in] ULONG ulStatusCode,
                        [in] LPCWSTR szStatusText);
        HRESULT OnDataAvailable([in] DWORD grfBSC, [in] DWORD dwSize,
                        [in] FORMATETC* pformatetc, [in] STGMEDIUM* pstgmed);
        HRESULT OnObjectAvailable( [in] REFIID riid, [in] IUnknown *punk);
        HRESULT OnLowResource([in] DWORD dwReserved);
        HRESULT OnStopBinding([in] HRESULT hrStatus, [in] LPCWSTR szStatusText);
        };

interface IPersistMoniker : IPersist {
        HRESULT IsDirty(void);
        HRESULT Load([in] BOOL fFullyAvailable, [in] IMoniker* pmkSrc, [in] IBindCtx* pbc, [in] DWORD grfMode);
        HRESULT Save([in] IMoniker* pmkDst, [in] IBindCtx* pbc, [in] BOOL fRemember);
        HRESULT SaveCompleted([in] IMoniker* pmkNew, [in] IBindCtx* pbc);
        HRESULT GetCurMoniker([out] IMoniker** ppmkCur);
        };

// IID_IAsyncMoniker: {660658f0-2e14-11cf-80fe-00aa00389b71}
DEFINE_GUID(IID_IAsyncMoniker, 0x660658f0, 0x2e14, 0x11cf, 0x80, 0xfe, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71);

HRESULT     CreateAsyncBindCtx( [in] DWORD dwReserved, [in] IBindStatusCallback* pbsc,
                        [in] IEnumFORMATETC* penumfmtetc, [out] IBindCtx** pbc);
HRESULT     RegisterBindStatusCallback([in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc,
                        [in] IBindStatusCallback** ppBSCBPrev, [in] DWORD dwReserved);
HRESULT     RevokeBindStatusCallback([in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc);
HRESULT     IsAsyncMoniker([in] IMoniker* pmk);
```

BINDVERB Enumeration

Values from the BINDVERB enumeration are passed to the client within IBindStatusCallback::GetBindInfo to distinguish different types of bind operations.

Value	Description
BINDVERB_GET	Perform a "get" operation (the default). The stgmedData member of the BINDINFO should be set to TYMED_NULL.
BINDVERB_POST	Perform a "post" operation. The data to post should be specified in the stgmedData member of the BINDINFO.
BINDVERB_PUT	Perform a "put" operation. The data to put should be specified in the stgmedData member of the BINDINFO.
BINDVERB_CUSTOM	Perform a custom operation (protocol specific, see szCustomVerb member of BINDINFO). The data to use should be specified in the stgmedData member of the BINDINFO.

BINDINFOF Enumeration

Values from the BINDINFOF enumeration are passed to the client within IBindStatusCallback::GetBindInfo to specify additional flags for the bind operation.

Value	Description
BINDINFOF_URLENCODESTGMEDDATA	Use URL encoding to pass is the data provided in the stgmedData member of the BINDINFO. (for PUT and POST operations)
BINDINFOF_URLENCODEEXTRAINFO	Use URL encoding to pass is the data provided in the szExtraInfo member of the BINDINFO.

BINDINFO Structure

The BINDINFO structure is returned to the asynchronous moniker through IBindStatusCallback::GetBindInfo. The user of the asynchronous moniker uses this structure to qualify the binding operation that will be occurring. The meaning of this structure is somewhat specific to the type of the asynchronous moniker. The technical specification provided applies here describes the meaning of the structure when used for URL monikers.

Member	Type	Description
cbSize	ULONG	Size of this structure, in bytes.
szExtraInfo	LPWSTR	The behavior of this field is moniker-specific. For URL monikers, this string is appended to the URL when the bind operation is started. Note: like all other OLE strings, this is a Unicode string that the client should allocate using CoTaskMemAlloc. The URL Moniker will free the memory later..
stgmedData	STGMEDIUM	Data to be PUT or POST.
grfBindInfoF	DWORD	Flag from the BINDINFOF enumeration specifying additional flags modifying the bind operation. (URL specific)
dwBindVerb	DWORD	A value from the BINDVERB enumeration specifying the action to be performed for the bind operation.
szCustomVerb	LPWSTR	String specifying a protocol specific custom verb to be used for the bind operation (only if grfBindInfoF is set to BINDINFOF_CUSTOM)
cbstgmedData	DWORD	Size of the data provided in stgmedData.

BINDF Enumeration

Values from the BINDF enumeration are returned to the binding layer from the client's IBindStatusCallback::OnStartBinding. These values are used to identify what type of binding the client wants from the moniker.

Value	Description
BINDF_ASYNCHRONOUS	The moniker should return immediately from IMoniker::BindToStorage or IMoniker::BindToObject. The actual result of the object bind or the data backing the storage will arrive asynchronously in calls to IBindStatusCallback::OnDataAvailable or IBindStatusCallback::OnObjectAvailable. If the client does not choose this flag, the bind operation will be synchronous, and the client will not receive any data from the bind operation until the IMoniker::BindToXXX call returns.
BINDF_ASYNCSTORAGE	The client of IMoniker::BindToStorage prefers that the IStorage and IStream objects returned in IBindStatusCallback::OnDataAvailable return E_PENDING when they reference data not yet available through I/O methods, rather than blocking until the data becomes available. This flag applies only to BINDF_ASYNCHRONOUS operations.[4]
BINDF_PULLDATA[5]	When this flag is specified, the asynchronous moniker will allow the client of IMoniker::BindToStorage to *drive* the bind operation by pulling the data, (rather than having the moniker driving the operation and pushing the data upon the client). Specifically, when this flag is chosen, *new data will only be read/downloaded after the client finishes reading all data that is currently available.* This means data will only be downloaded for the client after the client does an IStream::Read operation that blocks or returns E_PENDING. When the client chooses this flag, it must be sure to read all the data it can, even data that is not necessarily available yet. When this flag is not specified, the moniker will continue downloading data and will call the client with IBindStatusCallback::OnDataAvailable whenever new data is available. This flag applies only to BINDF_ASYNCHRONOUS bind operations.
BINDF_GETNEWESTVERSION	The moniker bind operation should retrieve the newest version of the data/object possible[6].
BINDF_NOWRITECACHE	The moniker bind operation should not store retrieved data in the disk cache.
BINDF_IGNORESECURITYPROBLEM	The moniker should ignore security problems during the bind operation. For examples of security problems, see the URL Monikers documentation for IHttpSecurity. This flag is dangerous and should only be set by clients when certain that this is ok - e.g. if a user claimed that all security problems should be ignored.

BSCF Enumeration

Values from the BSCF enumeration are passed to the client in IBindStatusCallback::OnDataAvailable to clarify the type of data which is available.

[4] Note: Asynchronous IStream objects will return E_PENDING while data is still downloading return S_FALSE for end-of-file.
[5] This flag was called BINDF_NOCOPYDATA in earlier versions of this document.
[6] For URL Monikers, this maps to an HTTP If-modified-since request. Cached data is only used if it is the most recent version.

Value	Description
BSCF_FIRSTDATANOTIFICATION	Identifies the first call to IBindStatusCallback::OnDataAvailable for a given bind operation.
BSCF_LASTDATANOTIFICATION	Identifies the last call to IBindStatusCallback::OnDataAvailable for a bind operation. t.
BSCF_INTERMEDIATEDATANOTIFICATION	Identifies an intermediate call to IBindStatusCallback::OnDataAvailable for a bind operation.

BINDSTATUS Enumeration

A single value from the BINDSTATUS enumeration is passed as ulStatusCode to the IBindStatusCallback::On-Progress function to tell the client about the progress of the bind operation.

Value	Description
BINDSTATUS_FINDINGRESOURCE	The bind operation is finding the resource that holds the object or storage being bound to. The szStatusText accompanying IBindStatus-Callback::OnProgress() provides the display name of the resource being searched for (e.g. “www.microsoft.com”).
BINDSTATUS_CONNECTING	The bind operation is connecting to the resource that holds the object or storage being bound to. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the resource being connected to (e.g. an IP address).
BINDSTATUS_SENDINGREQUEST	The bind operation is requesting the object or storage being bound to. The szStatusText accompanying IBindStatusCallback::On-Progress() provides the display name of the object (e.g. a filename).
BINDSTATUS_REDIRECTING	The bind operation has been redirected to a different data location. The szStatusText accompanying IBindStatusCallback::On-Progress() provides the display name of the new data location.
BINDSTATUS_USINGCACHEDCOPY	The bind operation is retrieving the requested object or storage from a cached copy. The szStatusText accompanying IBindStatusCall-back::OnProgress() is NULL.
BINDSTATUS_BEGINDOWNLOADDATA	The bind operation has begun receiving the object or storage being bound to. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the data location.
BINDSTATUS_DOWNLOADINGDATA	The bind operation continues to receive the object or storage being bound to. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the data location.
BINDSTATUS_ENDDOWNLOADDATA	The bind operation has finished receiving the object or storage being bound to. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the data location.

1.2.2BINDSTATUS_CLASSIDAVAILABLE For BindToObject() operations only - this notification is provided just before CoCreateInstance is called to create an instance of the object being bound to.The szStatusText accompanying IBindStatusCallback::OnProgress() provides the clsid of the new object in string format, allowing the client an opportunity to cancel the bind operation via IBinding::Abort, if desired.IBinding Interface

Asynchronous monikers must provide a *binding object* to their callers via IBindStatusCallback::OnStartBinding to allow control over the binding process.

When to Implement
Implementations of custom asynchronous monikers will implement and expose this interface to allow control of the bind operation. They will usually implement this interface on a separate object created by the moniker on a per-bind basis.

When to Use
Clients of asynchronous monikers obtain this interface through IBindStatusCallback::OnStartBinding. They typically keep a reference to this interface to be able to control the bind operation over the course of the bind or to receive protocol-specific information about the outcome of a bind operation.

IBinding::Abort

HRESULT IBinding::Abort();

Permanently aborts the bind operation. After aborting the bind operation the client may still receive some notifications about the binding.

An aborted bind operation will either result in a call to IBindStatusCallback::OnStopBinding with the error code E_ABORT, or a failure of the IMoniker::BindToObject/BindToStorage call in case this call did not previously return. At this point the bind operation is officially complete and the client must release any pointers obtained during the binding.

NOTE: Calling the last IBinding::Release does not terminate the bind operation.

Argument	Type	Description
Returns	S_OK	Success.
	S_FALSE	The bind operation was already aborted.
	E_FAIL	The bind operation could not be aborted.

IBinding::Suspend

HRESULT IBinding::Suspend();

Suspends the bind operation. The bind operation will be suspended until resumed by a later call to IBinding::Resume or canceled by a call to IBinding::Abort.

After calling IBinding::Suspend the client may still receive some notifications about the bind.

Argument	Type	Description
Returns	S_OK	Success.
	S_FALSE	The bind operation was already suspended.
	E_FAIL	The bind operation could not be suspended.

IBinding::Resume

HRESULT IBinding::Resume();

Resumes a suspended bind operation. The bind operation must have been previously suspended by a call to IBinding::Suspend.

Argument	Type	Description
Returns	S_OK	Success.
	S_FALSE	The bind operation was not previously suspended.
	E_FAIL	The suspended bind operation could not be resumed.

IBinding::SetPriority

HRESULT IBinding::SetPriority(nPriority);

Establishes the priority for the bind operation to nPriority. Priority values are taken from the Win32 thread priority APIs (SetThreadPriority and GetThreadPriority). The final priority is determined from values gathered from all clients of the bind operation.[7]

Argument	Type	Description
nPriority	LONG	A value indicating the priority to establish for this binding relative to other bindings and the system.
Returns	S_OK	Success.
	E_FAIL	The priority could not be changed.

[7] This method is currently unimplemented, and the policy for determining priority level is TBD. Tentatively, this policy may be to use the minimum priority from values specified by all clients.

IBinding::GetPriority

HRESULT IBinding::GetPriority(pnPriority);

This method retrieves the current priority of this bind operation. Priority values are taken from the Win32 thread priority APIs (SetThreadPriority and GetThreadPriority).

Argument	Type	Description
pnPriority	LONG*	Location to return a value indicating the priority established for this binding relative to other bindings and the system. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	The pnPriority argument is invalid.

IBinding::GetBindResult

HRESULT IBinding::GetBindResult(pclsidProtocol, pdwBindResult, pszBindResult, dwReserved);

This method can be used to query the protocol-specific outcome of a bind operation. It is typically called by an asynchronous moniker client upon receiving the IBindStatusCallback::OnStopBinding notification.

Argument	Type	Description
pclsidProtocol	CLSID *	A CLSID is returned here to identify the specific protocol that was used in the bind operation. For example, this may be one of CLSID_HttpProtocol, CLSID_FtpProtocol, CLSID_GopherProtocol, CLSID_HttpSProtocol, or CLSID_FileProtocol.
pdwBindResult	DWORD *	A protocol-specific DWORD bind result.
pszBindResult	LPWSTR *	A protocol-specific string describing the bind result. May be NULL.
dwReserved	DWORD	Reserved - must set to NULL.
Returns	S_OK	Success.
	E_INVALIDARG	The argument is invalid.

1.2.3IBindStatusCallback Interface

Clients have to provide an object exposing the following callback interface. An Asynchronous Moniker will provide information regarding the bind by calling methods on this interface.

This interface also serves the purpose of passing additional bind information to the moniker. The moniker will call two methods (GetBindInfo and GetPriority) after receiving the call to BindToObject/BindToStorage, to obtain this additional bind information. The IBindStatusCallback::QueryInterface method provides extensibility to IBindStatusCallback because it allows the moniker to request additional interfaces from the client in case the bind operation requires additional information or negotiaion

All methods in IBindStatusCallback may be invoked from within the IMoniker::BindToObject/BindToStorage call and after the moniker returns the call, if the bind information indicates asynchronous binding (BINDF_ASYNCHRONOUS).

Clients of asynchronous monikers register their callback interface into the bind context using RegisterBindStatusCallback. This API allows only one client to register callback functions for a single bind operation. If it is attempted to register more than one callback on a given bind context, the RegisterBindStatusCallback API will forcibly revoke the previous IBindStatusCallback and register the new one..

The moniker retrieves the IBindStatusCallback interface from the bind context.

If the Asynchronous Moniker needs to invoke other monikers as part of the bind operation, it may register its own IBindStatusCallback interface in the bind context just like any other client of a moniker would. This allows chaining of notifications, where the “out-most” moniker can consolidate its own progress with the inner moniker’s progress notifications.

When to Implement

Any client of an Asynchronous Moniker has to implement this interface, in order to obtain asynchronous behavior. A client will implement this interface on a separate object (similar to a site object) that it associates with a specific bind operation. **Note: A client should only implement those callback member functions that it is interested in. Almost all the callback functions are optional, and a client may return NOERROR in most cases as noted below.**

The methods in IBindStatusCallbacks do not provide information as to which specific bind the notification belongs to, so that a client will want to provide a separate object instance for each simultaneous asynchronous bind operation it initiates.

The client registers the interface into the bind context by calling RegisterBindStatusCallback. The bind context will keep a reference to the object. The moniker will optionally add references to this object.

When to Use

Implementations of Asynchronous Monikers will use this interface for two purposes:

- Obtain additional bind information: During the call to BindToStorage/BindToObject the moniker will call IBindStatusCallback::GetBindInfo to check at least the BINDF_ASYNCHRONOUS flag. If this flag is not set, the method may not return until the object/storage object is available. The moniker may call IBindStatusCallback::GetPriority at this time or at a later point. Lastly, the moniker may call IBindStatusCallback::QueryInterface to request a new interface from the client if the bind operation needs further information or additional services from the client.
- Provide notifications: During the call to BindToStorage/BindToObject the moniker may call any of the notification methods as well as the bind information methods. After returning from the call, the moniker will provide additional notifications for the duration of the bind.

IBindStatusCallback::QueryInterface

HRESULT IBindStatusCallback::QueryInterface(riid, ppvObject);

The moniker calls this method to query the client for additional services necessary for completing the bind operation. This mechanism provides extensibility to the IBindStatusCallback interface, because it allows querying the client for new callback interfaces for passing information or querying information. A moniker client may also provide these "extension" callback interfaces via an IServiceProvider interface. After the moniker uses IBindStatusCallback::QueryInterface to directly query the client for an extension interface, the moniker will then query for the IServiceProvider interface, and will then try using IServiceProvider::QueryService to query for the desired extension interface.[8]

For examples of additional services that may be requested from an asynchronous moniker, see the URL Monikers specification.

Argument	Type	Description
riid	REFIID	The REFIID for the interface for the requested service.
ppvObject	void *	The interface returned by the client
Returns	S_OK	Success. The interface returned is used by the moniker to communicate further information pertaining to the bind operation.
	E_NOINTERFACE	The client does not know how to support the requested interface. Note: if none of the callbacks registered for a particular bind operation return S_OK to this call, the bind operation will perform default action.
	E_OUTOFMEMORY	Out of memory.
	E_INVALIDARG	One or more arguments are invalid.

[8] Because IServiceProvider::QueryService is not restricted by COM identity rules in the same way as QueryInterface, this mechanism allows moniker clients to delegate such extension services to other objects. Note that if delegating a QueryService request to another IBindStatusCallback implementation, one should first delegate an interface acquired via QueryInterface before querying the secondary IBindStatusCallback implementation for IServiceProvider.

IBindStatusCallback::GetBindInfo

HRESULT IBindStatusCallback::GetBindInfo(pgrfBINDF, pbindinfo);

An asynchronous moniker calls this method to obtain the bind information for the bind operation. The moniker calls this method within its implementations of BindToObject and BindToStorage before returning, in order to obtain information about the specific bind operation. Note that even though multiple IBindStatus-Callback interfaces may be registered on the BindCtx, ***only one*** moniker client will actually receive the IBind-StatusCallback::GetBindInfo callback. (see additional details under RegisterBindStatusCallback).

Argument	Type	Description
pgrfBINDF	DWORD*	Location to return a value taken from the BINDF enumeration which indicates whether the bind should proceed synchronously or asynchronously.
pbindinfo	BINDINFO*	Location to return the BINDINFO structure which describes how the caller wants the binding to occur.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IBindStatusCallback::OnStartBinding

HRESULT IBindStatusCallback::OnStartBinding(dwReserved, pbinding);

Asynchronous monikers typically call this method while initiating the bind operation, during IMoniker::BindToStorage or IMoniker::BindToObject.

This notification passes the binding object associated with the current bind operation to the client. The binding object allows control of the bind operation and the client may call it at any time.

To keep a reference to the binding object, the client must store the pointer and call AddRef, and Release it when they are done. Calling Release does not cancel the bind operation, it simply frees the reference to the IBinding interface sent in the callback.

Note: if a client is not interested in this callback, it may simply return from this callback with a simple return value of S_OK *or* E_UNIMPL.

Argument	Type	Description
dwReserved	DWORD	Reserved, must be zero..
pbinding	IBinding*	The IBinding interface of the current bind operation. May not be NULL. The client should call AddRef() on this pointer if it wishes to keep a reference to the binding object.*Returns* S_OK Success.
Returns	S_OK	Success.
	E_INVALIDARG	The pbinding argument is invalid.

IBindStatusCallback::GetPriority

HRESULT IBindStatusCallback::GetPriority(pnPriority);

The moniker calls this method, typically prior to initiating the bind operation, to obtain the priority for the bind. This method may be called at any time during the bind operation if the moniker needs to make new priority decisions.[9]

It may use this value for setting the actual priority of a thread associated with a download operation but more commonly it will interpret the priority to perform its own scheduling among multiple bind operations. The moniker must not change the priority of the thread used for calling BindToStorage or BindToObject.

Note: if a client is not interested in this callback, it may simply return from this callback with a simple return value of S_OK *or* E_UNIMPL.

[9] Policy for determining priority level is TBD.

Argument	Type	Description
pnPriority	LONG*	Location to return a value indicating the priority of this download. Priorities may be any of the constants defined for prioritizing threads. See the Win32 documentation for SetThreadPriority and GetThreadPriority for details.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IBindStatusCallback::OnProgress

HRESULT IBindStatusCallback::OnProgress(ulProgress, ulProgressMax, ulStatusCode, szStatusText);

The moniker calls this method repeatedly to indicate the current progress of this bind operation, typically at reasonable intervals during a lengthy operation.

The client may use the progress notification to provide progress information to the user (ulProgress/ulProgressMax and szStatusText) or to make programmatic decisions based on ulStatusCode.

Note: if a client is not interested in this callback, it may simply return from this callback with a simple return value of S_OK *or* E_UNIMPL.

Argument	Type	Description
ulProgress	ULONG	Indicates the current progress of the bind operation relative to the expected maximum indicated in ulProgressMax.
ulProgressMax	ULONG	Indicates the expected maximum value of ulProgress for the duration of calls to OnProgress for this operation. Note that this value may change across invocations of this method.
ulStatusCode	ULONG	Provides additional information regarding the progress of the bind operation. Valid values are taken from the BINDSTATUS enumeration.
szStatusText	LPCWSTR	Information about the current progress, depending on the value of ulStatusCode. See definition of the BINDSTATUS enumeration for further details.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IBindStatusCallback::OnDataAvailable

HRESULT IBindStatusCallback::OnDataAvailable(grfBSCF, dwSize, pfmtetc, pstgmed);

During asynchronous IMoniker::BindToStorage() bind operations, an asynchronous moniker calls this method to provide data to the client as it becomes available. Note that the behavior of the storage returned in pstgmed depends on the BINDF flags returned in IBindStatusCallback::GetBindInfo—this storage may be asynchronous or blocking, and the bind operation may follow a “datapull” model or a “datapush” model.. Furthermore, it is important to note that *for BINDF_PULLDATA bind operations, it is not possible to seek backwards* into data streams provided in IBindStatusCallback::OnDataAvailable. On the other hand, for push model bind operations, it is commonly possible to seek back into a data stream and read any data that has been downloaded for an ongoing IMoniker::BindToStorage operation.

Argument	Type	Description
grfBSCF	DWORD	Values taken from the BSCF enumeration.
dwSize	DWORD	The amount (in bytes) of total data available from the current bind operation.
pfmtetc	FORMATETC*	Indicates the format of the available data when called as a result of IMoniker::BindToStorage. If there is no format associated with the available data, pformatetc may contain CF_NULL.
pstgmed	STGMEDIUM*	Holds the actual data that became available when called as a result of IMoniker::BindToStorage. If it wishes to keep the data in pstgmed alive, the client should call AddRef() on pstgmed->pUnkForRelease (if the pointer is non-NULL), and eventually use the ReleaseStgMedium API to release the storage.[10]
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IBindStatusCallback::OnObjectAvailable

HRESULT IBindStatusCallback::OnObjectAvailable(riid, punk);

During asynchronous IMoniker::BindToObject() bind operations, an asynchronous moniker calls this method to pass the requested object interface pointer to the client. This method is never called for IMoniker::BindToStorage operations.

Argument	Type	Description
riid	REFIID	The REFIID of the requested interface.
punk	IUnkown *	The object pointer requested in the call to IMoniker::BindToObject. The client should call AddRef() on this pointer in order to maintain a reference to the object.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IBindStatusCallback::OnLowResource

HRESULT IBindStatusCallback::OnLowResource(dwReserved);

The moniker calls this method when it detects low resources. The client should free any resource it no longer needs when receiving this notification.

Note: if a client is not interested in this callback, it may simply return from this callback with a simple return value of S_OK *or* E_UNIMPL.

Argument	Type	Description
dwReserved	DWORD	Reserved for future use. Must be zero.
Returns	S_OK	Success.

IBindStatusCallback::OnStopBinding

HRESULT IBindStatusCallback::OnStopBinding(hrStatus, szStatusText);

The moniker calls this method to indicate the end of the bind operation. This method is *always* called, whether the bind operation succeeded, failed, or was aborted by a client. At this point, the client may use IBinding::GetBindResult to query protocol-specific information about the outcome of the bind operation. At the end of this callback, the moniker client must call Release() on the IBinding pointer it received in IBindStatusCallback::OnStartBinding.

[10] Note that keep the pstgmed alive will not always be possible, in which case pstgmed->pUnkForRelease will be NULL. For example, this will be the case when using URL Monikers to download data that is not being cached.

Note: if a client is not interested in this callback, it may simply return from this callback with a simple return value of S_OK *or* E_UNIMPL. However, the client must implement this callback if it has kept a reference count onto the IBinding object provided in IBindStatusCallback::OnStartBinding.

Argument	Type	Description
hrStatus	HRESULT	Status code which would have been returned from the method that initiated the bind operation (IMoniker::BindToObject or IMoniker::BindToStorage).
szStatusText	LPCWSTR	Status text. In case of error, this string may provide additional information describing the error[11]. In case of success, szStatusText provides the friendly name of the data location bound to.
Returns	S_OK	Success.

1.2.4IPersistMoniker Interface

Objects, especially *asynchronous-aware* objects may expose the IPersistMoniker interface to obtain more control over the way they bind to their persistent data.

Existing moniker implementations QueryInterface the client for persistence interfaces such as IPersistFile, IPersistStream[Init], or IPersistStorage as part of their BindToObject implementation as they are instantiating and initializing the object. IPersistMoniker allows moniker implementations and other applications which instantiate objects from persistent data to give control to the object to choose how it wishes to bind to its persistent data. A typical usage scenario for objects is to implement IPersistMoniker::Load by calling IMoniker::BindToStorage for the interface they prefer – IStorage, IStream, asynchronously bound, etc.

Unlike some other persistent object interfaces IPersistMoniker does not include an InitNew method. This means that IPersistNew cannot be used to initialize an object to a 'freshly initialized state'. Clients of IPersistNew, who wish to initialize the object should QueryInterface for IPersistStreamInit, IPersistMemory, and/or IPersistPropertyBag and use the InitNew method found there to initialize the object. The client then can safely use IPersistMoniker to save the persistent state of the object.

When to Implement

Implement IPersistMoniker on any object which can persist to multiple storage mediums or which can take advantage of any of the asynchronous stream, storage, or IMoniker::BindToStorage behavior described above.

When to Use

Custom moniker implementations should support IPersistMoniker as the highest or most flexible persistence interface in their implementation of IMoniker::BindToObject if they are instantiating and arbitrary class and need to initialize it from persistent data. Typically these monikers should use the published persistence interfaces in the following order: IPersistMoniker, IPersistStream[Init], IPersistStorage, IPersistFile, IPersistMemory.

IPersistMoniker::IsDirty

HRESULT IPersistMoniker::IsDirty();

Checks an object for changes since it was last saved.

Argument	Type	Description
Returns	S_OK	Yes, the object has changed since it was last saved.
	S_FALSE	No, the object has not changed since it was last saved.

IPersistMoniker::Load

HRESULT IPersistMoniker::Load(fFullyAvailable, pmkSrc, pbc, grfMode);

Loads an object from the persistent state referred to by pmkSrc. Typically the object will immediately bind to its persistent state using pmkSrc->BindToStorage(pbc, ...) for either IStream or IStorage.

[11] Currently the string is empty in error cases.

Argument	Type	Description
fFullyAvailable	BOOL	If TRUE, then the data referred to by the moniker has already been loaded once, and subsequent binding to the moniker should be synchronous. If this value is FALSE, then the implementation of Load should be prepared to bind to the moniker asynchronously.
pmkSrc	IMoniker*	A reference to the persistent state to initialize this object from.
pbc	IBindCtx*	The bind context to use for any moniker binding during this method.
grfMode	DWORD	A combination of the values from the STGM enumeration which indicate the access mode to use when binding to the persistent state. The IPersistMoniker::Load method can treat this value as a suggestion, adding more restrictive permissions if necessary. If grfMode is zero, the implementation should bind to the persistent state using default permissions.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IPersistMoniker::Save

HRESULT IPersistMoniker::Save(pmkDst, pbc, fRemember);

Requests that the object save itself into the location referred to by pmkDst.

Argument	Type	Description
pmkDst	IMoniker*	Moniker to the location where the object should persist itself. The object typically binds to the location using pmkDst->BindToStorage for either IStream or IStorage. May be NULL, in which case the object is requested to save itself to the same location referred to by the moniker passed to it in IPersistMoniker::Load. This may act as an optimization to prevent the object from binding, since it has typically already bound to the moniker it was loaded from.
pbc	IBindCtx*	The bind context to use for any moniker binding during this method.
fRemember	BOOL	Indicates whether pmkDst is to be used as the reference to the current persistent state after the save. If TRUE, pmkDst becomes the reference to the current persistent state and the object should clear its dirty flag after the save. If FALSE, this save operation is a "Save A Copy As ..." operation. In this case, the reference to the current persistent state is unchanged and the object should not clear its dirty flag. If pmkDst is NULL, the implementation should ignore the fRemember flag..
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IPersistMoniker::SaveCompleted

HRESULT IPersistMoniker::SaveCompleted(pmkNew, pbc);

Notifies the object that it has been completely saved and points it to its new persisted state. Typically the object will immediately bind to its persistent state using pmkNew->BindToStorage(pbc, ...) for either IStream or IStorage, as in IPersistMoniker::Load.

Argument	Type	Description
pmkNew	IMoniker*	The moniker to the object's new persistent state, or NULL as an optimization if the moniker to the object's new persistent state is the same as the previous moniker to the object's persistent state – only allowed if there was a prior call to IPersistMoniker::Save with fRemember=TRUE – in which case the object need not rebind to pmkNew.
pbc	IBindCtx*	The bind context to use for any moniker binding during this method.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IPersistMoniker::GetCurMoniker

HRESULT IPersistMoniker::GetCurMoniker(ppmkCur);

Retrieves the moniker that refers to the object's persistent state. Typically, this is the moniker last passed to the object via IPersistMoniker::Load or IPersistMoniker::Save/::SaveCompleted.

Argument	Type	Description
ppmkCur	IMoniker**	Location to return the moniker to the object's current persistent state.
Returns	S_OK	Success.
	E_INVALIDARG	The ppmkCur argument is invalid.

1.2.5Helper Functions

CreateAsyncBindCtx

HRESULT CreateAsyncBindCtx(dwReserved, pbsc, penumfmtetc, ppbc);

Creates an asynchronous bind context for use with asynchronous monikers.

This function automatically registers the IBindStatusCallback and the IEnumFORMATETC interfaces with the bind context.

CreateAsyncBindCtx also initializes the grfFlags in the bind context's BIND_OPTS to the value of BIND_MAY-BOTHERUSER (using IBindCtx::SetBindOptions). This means that by default a moniker bind operation using asynchronous bind contexts may try to interrupt the user in order to display UI to complete the bind oper-ation.[12]

Argument	Type	Description
dwReserved	DWORD	Reserved for future use. Must be zero.
pbsc	IBindStatusCallback*	The callback to receiving data availability and progress notifica-tion.
penumfmtetc	IEnumFORMATETC*	Enumerator of formats to use for format negotiation during bind-ing, if applicable. May be NULL, in which case the caller is not in-terested in format negotiation during binding and the default for-mat of the object will be bound to.
ppbc	IBindCtx**	Location to return the new bind-context.
Returns	S_OK	Success.
	E_OUTOFMEMORY	Out of memory.

[12] Such interruptions are necessary for actions such as HTTP authentication. A client may reset this flag to 0 if they wish to run bind operations in UI-free mode.

	E_INVALIDARG	One or more arguments are invalid.

RegisterBindStatusCallback

HRESULT RegisterBindStatusCallback(pbc, pbsc, ppBSCBPrev, dwReserved);

Registers an IBindStatusCallback with an existing bind context. The given IBindStatusCallback will be the only callback interface to receive asynchronous notifications during the bind operation, although the callback may choose to delegate notifications on to other callbacks. This callback interface will be used for all asynchronous bind operations using this bind context, until either the callback interface is revoked via RevokeBindStatusCallback, or until the bind context is destroyed.

Note also that this function *revokes* and *returns* the previous IBindStatusCallback registered on the BindCtx.

Argument	Type	Description
pbc	IBindCtx*	The bind context to register the callback with.
pbsc	IBindStatusCallback*	The callback interface to register.
ppBSCBPrev	IBindStatusCallback**	**Optional** pointer. If set to a non-NULL address, the previous IBindStatusCallback registered on the BindCtx will be revoked, AddRef()ed, and returned in this parameter.
dwReserved	DWORD	Reserved for future extension.
Returns	S_OK	Success.
	E_OUTOFMEMORY	Insufficient memory to register the callback with the bind context.
	E_INVALIDARG	One or more arguments are invalid.

RevokeBindStatusCallback

HRESULT RevokeBindStatusCallback();

Revokes an IBindStatusCallback callback previously registered on a bind context. This call will not succeed if it is made during a bind operation. Note: it is not necessary to make this call for every use of a bind context—it is technically possible (although not recommended) to reuse the same bind context and callback object for many bind operations. Upon calling IBindCtx::Release(), all registered object on that bind context will be revoked, including the IBindStatusCallback. Therefore releasing a bind context implicitly releases all registered IBindStatusCallback objects. However, if one chooses to reuse a bind context, one can use RevokeBindStatusCallback to remove a registered IBindStatusCallback object so it is not re-used.

Argument	Type	Description
pbc	IBindCtx*	The bind context to revoke the callback from.
pbsc	IBindStatusCallback*	The callback interface to revoke.
Returns	S_OK	Success.
	E_FAIL	The IBindStatusCallback is not registered on the bind context.
	E_INVALIDARG	One or more arguments are invalid.

IsAsyncMoniker

HRESULT IsAsyncMoniker(pmk);

Tests if a moniker supports asynchronous binding. A moniker implementation indicates that it is asynchronous by supporting the IMonikerAsync interface, an "empty" interface which is actually just IUnknown, as demonstrated here:

```
 STDMETHODIMP
MyCustomMoniker::QueryInterface(REFIID riid, void** ppv) {
        if (riid == IID_IUnknown || riid == IID_IPersistStream || riid == IID_IMoniker || riid == IID_IAsyncMoniker) {
            *ppv = this;
            AddRef();
            return S_OK;
            }
        *ppv = NULL;
        return E_NOINTERFACE;
    }
```
IsAsyncMoniker tests support for this interface and also handles composite monikers correctly.

Argument	Type	Description
pmk	IMoniker*	The moniker to test.
Returns	S_OK	Yes, the moniker is asynchronous.
	S_FALSE	No, the moniker is not asynchronous.
E_INVALIDARG	The pmk argument is invalid.	