Legal Information

ActiveX Software Development Kit

This document is an early release of the final documentation. It is meant to specify and accompany software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of the final specification or software. Microsoft assumes no responsibility for any damages that might occur either directly or indirectly from these inaccuracies.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights except as expressly provided in any written license agreement from Microsoft.

Legal Information

ActiveX Software Development Kit

This document is an early release of the final documentation. It is meant to specify and accompany software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of the final specification or software. Microsoft assumes no responsibility for any damages that might occur either directly or indirectly from these inaccuracies.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights except as expressly provided in any written license agreement from Microsoft.

IAsyncMoniker

This interface is simply an implementation of **IUnknown**. There are no additional methods. It is is used to allow clients to determine if a moniker supports asynchronous binding.

When to Implement

Implement this interface on any asynchronous moniker so clients can determine that the moniker supports asynchronous binding.

When to Use

QueryInterface for **IAsyncMoniker** to determine if the moniker supports asynchronous binding.

Methods in Vtable Order

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

See Also

IAuthenticate

During a bind operation, the moniker can obtain some additional bind information by calling the bind client's **IBindStatusCallback::GetBindInfo**. In some cases, additional callback interfaces can be provided by the client requesting the bind operation to supply even more services to the moniker performing the bind. These additional interfaces are typically requested by the moniker calling **IBindStatusCallback::QueryInterface** for the interface desired. However, a moniker client can also provide additional interfaces through **IServiceProvider**.

In the case of URL monikers, the **IAuthenticate** interface is an additional callback interface that the client can provide.

When to Implement

Clients of URL monikers can implement this interface to provide the URL moniker with information to authenticate the user. The moniker client that implements **IAuthenticate** can display a custom user interface or use a custom password list to specify a username and password for accessing secure information on Internet resources.

When to Use

The URL moniker calls this interface when the bind operation requires authentication for an URL download. To get a pointer to this interface, call **IBindStatusCallback::QueryInterface**.

Methods in Vtable Order

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

IAuthenticate Methods	Description
Authenticate	Supplies basic authentication to an URL moniker from a bind client.

See Also

IHttpNegotiate, **IServiceProvider**, IWindow, IWinINetInfo

IAuthenticate::Authenticate

Called by the URL Moniker when it needs basic authentication information from a bind client.

```
HRESULT Authenticate(
    HWND *phwnd,            //Pointer to HWND
    LPWSTR *pszUserName,    //Username for authentication.
    LPWSTR *pszPassword     //Password for authentication.
  );
```

Parameter

phwnd

[out] Pointer to a client-provided HWND of the parent window for a default authentication user interface. If no user interface is desired, the client must provide a username and password in the other parameters, and this handle.is set to the value INVALID_HANDLE_VALUE.

pszUserName

[out] Client-provided username for authentication. If the client returns a value here, it should also set **phwnd* = INVALID_HANDLE_VALUE.

pszPassword

[out] Client-provided password for authentication. If the client returns a value here, it should also set **phwnd* = INVALID_HANDLE_VALUE.

Return Values

S_OK

The operation was successful.

E_INVALIDARG

One or more parameters are invalid.

Remarks

The client can return username and password strings, or it can provide an HWND that is used to present default authentication user interface.

IBindHost

This interface is implemented on a site object to supply another object with services and information it requires when performing an asynchronous data transfer, specifically parsing a name and converting it to a moniker and initializing a bind context.

Methods in Vtable Order

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

IBindHost Methods	Description
ParseDisplayName	Turns a text name into a moniker.
GetBindContext	Supplies a pointer to an implementation of **IBindCtx**.

IBindHost::GetBindCtx

IBindHost::GetBindCtx is the factory method for a host-initialized bind context. It supplies a pointer to an implementation of **IBindCtx**. Its arguments are identical to those for the API function **CreateBindCtx**.

```
HRESULT GetBindCtx(
    DWORD dwReserved,      //Reserved for future use; must be zero
    IBindCtx ** ppbc       //Indirect pointer to the IBindCtx interface
  );
```

Parameters

dwReserved

[in] Reserved for future use; must be zero.

ppbc

[out] Indirect pointer the **IBindCtx** pointer. The caller is responsible for calling **IBindCtx::Release** when the bind context is no longer needed.

Return Values

S_OK

The bind context was successfully obtained and the caller is responsible for the interface pointer.

E_OUTOFMEMORY

There is insufficient memory to create the bind context.

E_UNEXPECTED

An unknown error occurred.

Remarks

E_NOTIMPL is not allowed — a bind host is responsible for providing bind context creation services.

IBindHost::ParseDisplayName

Provides the caller with a means to turn some sort of text name into a moniker such that the caller does not have to interpret the name in any way itself. In many cases, the implementation of **IBindHost::ParseDisplayName** will simply call **MkParseDisplayNameEx**, but this method gives the implementor of **IBindHost** a chance to catch host-specific strings that **MkParseDisplayNameEx** would not otherwise recognize.

```
HRESULT ParseDisplayName(
    LPOLESTR   pszName,     //Pointer to the string to parse
    IMoniker ** ppmk        //Indirect pointer to the IMoniker interface for the new moniker
  );
```

Parameters

pszName
[in] Pointer to the string containing the name to parse.

ppmk
[out] Indirect pointer to **IMoniker** interface for the moniker created from *pszName*. The caller is responsible for calling **IMoniker::Release** when the moniker is no longer needed.

Return Values

S_OK
The moniker was successfully obtained and the caller is responsible for the interface pointer.

E_OUTOFMEMORY
There is insufficient memory to create the moniker.

E_UNEXPECTED
An unexpected error occurred.

MK_E_SYNTAX
The bind host was unable to parse the string into a moniker because the information in the name was unrecognizable.

Remarks

E_NOTIMPL is not allowed — a bind host is responsible for providing moniker parsing services.

IBinding

The **IBinding** interface provides methods that the client of an asynchronous moniker can call to control the progress of the bind operation. Asynchronous monikers call the client's IBindStatusCallback::OnStartBinding method to provide the client with a pointer to the **IBinding** interface.

When to Implement

Implementations of custom asynchronous monikers must provide this interface to allow control of the bind operation. Usually, this interface is implemented on a separate object created by the asynchronous moniker for a particular bind operation.

When to Use

Clients of asynchronous monikers obtain a pointer to this interface when the moniker calls the client's **IBindStatusCallback::OnStartBinding** method. They typically keep a reference to this interface to be able to control the progress of the bind operation or to receive protocol-specific information about the outcome of the bind operation.

Methods in Vtable Order

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

IBinding Methods	Description
Abort	Permanently aborts the bind operation
Suspend	Suspends the bind operation.
Resume	Resumes a suspended bind operation.
SetPriority	Establishes the priority for the bind operation.
GetPriority	Retrieves the current priority of this bind operation.
GetBindResult	Queries the protocol-specific outcome of a bind operation.

IBinding::Abort

Permanently ends the bind operation.

HRESULT Abort(void);

Return Values

S_OK
The bind operation was successfully aborted.

S_FALSE
The bind operation was already aborted.

E_FAIL
The bind operation could not be aborted.

Remarks

When a bind operation is aborted, it calls **IBindStatusCallback::OnStopBinding** with a corresponding error code or failure of the **IMoniker::BindToObject** or the **IMoniker::BindToStorage** call in case this call did not previously return.

After aborting the bind operation, the client must still release any pointers obtained during the binding, and the client may still receive some notifications about the binding.

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

See Also

IBindStatusCallback::OnStopBinding

IBinding::GetBindResult

Queries the protocol-specific outcome of a bind operation.

```
HRESULT GetBindResult(
    CLSID *pclsidProtocol,      //Pointer to the CLSID that identifies the protocol used
    DWORD *pdwBindResult,       //Pointer to bind result
    LPWSTR *pszBindResult,      //Pointer to protocol-specific bind result
    DWORD dwReserved            //Reserved for future use; must be NULL
  );
```

Parameters

pcslidProtocol
[out] Pointer to the **CLSID** variable for the protocol used.

pdwBindResult
[out] Pointer to the **DWORD** variable containing the protocol-specific bind result.

pszBindResult
[out] Pointer to the **LPWSTR** variable containing the protocol-specific bind result.

dwReserved
[in] Reserved for future use; must be NULL.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One of the parameters is not valid.

Remarks

This method is typically called by the client of an asynchronous moniker when the client's **IBindStatusCallback::OnStopBinding** method is called.

See Also

IBindStatusCallback::OnStopBinding

IBinding::GetPriority

Retrieves the current priority of this bind operation.

HRESULT GetPriority(
  LONG ****pnPriority //Pointer to the priority value
);

Parameters

pnPriority
  [out] Pointer to a **LONG** variable indicating the priority established for this binding relative to other bindings and the system. May not be NULL.

Return Values

S_OK
  The priority was successfully returned.

E_INVALIDARG
  The *pnPriority* parameter is invalid.

Remarks

Priority values are taken from the Win32 thread priority APIs (**SetThreadPriority** and **GetThreadPriority**).

See Also

IBinding::SetPriority

IBinding::Resume

Resumes a suspended bind operation.

HRESULT Resume(void);

Return Values

S_OK
: The resume operation was successful.

S_FALSE
: The bind operation was not previously suspended.

E_FAIL
: The suspended bind operation could not be resumed.

Remarks

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

See Also

IBinding::Suspend

IBinding::SetPriority

Establishes the priority for the bind operation.

HRESULT SetPriority(
  LONG *nPriority* //Relative priority for this binding
);

Parameters

nPriority
  [in] Priority to establish for this binding relative to other bindings and the system.

Return Values

S_OK
  The priority was successfully set.

E_FAIL
  The priority could not be changed.

Remarks

IBinding::SetPriority stablishes the priority for the bind operation. Priority values are taken from the Microsoft® Win32® thread priority APIs (**SetThreadPriority** and **GetThreadPriority**). The final priority is determined from values gathered from all clients of the bind operation.

Note that this method is currently unimplemented for URL monikers and the policy for determining the priority level is TBD.

See Also

IBinding::GetPriority

IBinding::Suspend

Suspends the bind operation.

HRESULT Suspend(void);

Return Values

S_OK
: The suspend operation was successful.

S_FALSE
: The bind operation was already suspended.

E_FAIL
: The bind operation could not be suspended.

Remarks

The bind operation will be suspended until resumed by a subsequent call to **IBinding::Resume** or canceled by a subsequent call to **IBinding::Abort**.

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

See Also

IBinding::Abort, IBinding::Resume

IBindStatusCallback

A client requesting an asynchronous bind operation must provide a notification object exposing the **IBindStatusCallback** interface. The asynchronous moniker provides information on the bind operation to the client by calling notification methods on the client's callback interface.

Besides providing notification methods, this interface also allows the client to pass additional bind information to the moniker. To obtain this additional bind information, the moniker calls two **IBindStatusCallback** methods (**GetBindInfo** and **GetPriority**) after receiving a call either to **IMoniker::BindToObject** or **IMoniker::BindToStorage**. **IBindStatusCallback::QueryInterface** 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** can be called from within **IMoniker::BindToObject** or **IMoniker::BindToStorage**. These methods can also be called after the moniker returns if the bind information indicates asynchronous binding (BINDF_ASYNCHRONOUS).

Clients of asynchronous monikers register their callback interface in the bind context by calling the RegisterBindStatusCallback function. Multiple clients can register callback interfaces for the same bind operation. Each client specifies the notifications it is interested in receiving in the *grfBSCOption* parameter.

During the bind operation, these callbacks are called in an arbitrary order, and the asynchronous moniker may set policy and limit certain callback notifications to only one of the registered callback interfaces. For example, the moniker may limit calls to the **GetBindInfo**, **OnDataAvailable**, or **OnObjectAvailable** methods to a single callback interface.

Note that, in the case of URL monikers, the policy is not documented and is likely to change. It is advised that only one registered callback interface (usually the last one registered) be responsible for the callbacks to the **GetBindInfo**, **OnDataAvailable**, and **OnObjectAvailable** methods.

When to Implement

Any client of an asynchronous moniker must implement this interface to obtain asynchronous behaviour. Typically, a client implements this interface on a separate object, similar to a site object, that it associates with a specific bind operation.

Note that most of the notification methods in this interface are optional. A client should implement only those notifications it is interested in receiving. The client can return E_UNIMP or S_OK for notification methods it does not wish to receive.

The methods in **IBindStatusCallback** do not identify the specific bind operation that the notification belongs to, so a client should provide a separate object instance for each simultaneous asynchronous bind operation it initiates.

When the client registers the interface in the bind context by calling RegisterBindStatusCallback, the bind context keeps a reference to the callback object. The moniker can optionally add references to this object.

When to Use

Implementations of asynchronous monikers will use this interface for two purposes:

- **Obtain additional bind information**. Throughout the bind operation, the moniker can call the client to obtain additional binding information. For example, in the implementation of **IMoniker::BindToStorage** or **IMoniker::BindToObject**, the moniker calls **IBindStatusCallback::GetBindInfo** to check at least the BINDF_ASYNCHRONOUS flag. If this flag is not set, the bind operation cannot return until the object or storage object is available. As another example, the moniker can call **IBindStatusCallback::GetPriority** during its

IMoniker::BindToStorage or **IMoniker::BindToObject** methods, or it can wait until a later point to get the priority of the bind operation. Also, the moniker can 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**. In the implementation of **IMoniker::BindToStorage** or **IMoniker::BindToObject**, the moniker can call any of the notification methods as well as the bind information methods. After returning from the call, the moniker can call notification methods to provide additional notification throughout the bind operation.

Methods in Vtable Order

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

IBindStatusCallback Methods	Description
GetBindInfo	Called by asynchronous moniker to get bind info
OnStartBinding	Tells the client which callback methods it is registered for receiving.
GetPriority	Gets data during asynchronous bind operations.
OnProgress	Indicates the current progress of this bind operation.
OnDataAvailable	Retrieves the current priority of this bind operation.
OnObjectAvailable	Called by asynchronous moniker to to pass the requested object interface pointer to the client.
OnLowResource	An asynchronous moniker calls this method when it detects low resources.
OnStopBinding	An asynchronous moniker calls this method to indicate the end of the bind operation.

IBindStatusCallback::GetBindInfo

An asynchronous moniker calls this method to obtain the bind information for the bind operation.

```
HRESULT GetBindInfo(
    DWORD *pgrfBINDF,       //Pointer to BINDF value
    BINDINFO *pbindinfo     //Pointer to BINDINFO structure
  );
```

Parameters

pgrfBINDF

[out] Pointer to a value taken from the BINDF enumeration indicating whether the bind should proceed synchronously or asynchronously.

pbindinfo

[in, out] Pointer to the BINDINFO structure which describes how the client wants the binding to occur.

Return Values

S_OK

The operation was successful.

E_INVALIDARG

One or more parameters are invalid.

Remarks

The moniker calls this method in its implementations of **IMoniker::BindToObject** and **IMoniker::BindToStorage** to obtain information about the specific bind operation. Note that even though multiple **IBindStatusCallback** interfaces can be registered in the bind context, only one moniker client actually receives the **IBindStatusCallback::GetBindInfo** callback.

See Also

BINDF, BINDINFO, RegisterBindStatusCallback

IBindStatusCallback::GetPriority

An asynchronous moniker calls this method to obtain the priority for the bind operation.

HRESULT GetPriority(
  LONG ****pnPriority //Pointer to priority value
);

Parameters

pnPriority
  [out] Pointer a **LONG** value indicating the priority of this bind operation. Priorities may be any of the constants defined for prioritizing threads. See the Win32 documentation for **SetThreadPriority** and **GetThreadPriority** for details.

Return Values

S_OK
  The operation was successful.

E_INVALIDARG
  The *pnPriority* parameter is invalid.

Remarks

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

The moniker can use this value for setting the actual priority of a thread associated with a bind operation but more commonly it will interpret the priority to perform its own scheduling among multiple bind operations. Note that the policy for determining priority for URL monikers is not yet determined. The moniker must not change the priority of the thread used for calling **IMoniker::BindToStorage** or **IMoniker::BindToObject**.

Notes to Implementers

A client can return E_UNIMPL or S_OK if it is not interested in receiving this notification.

See Also

IBindStatusCallback::OnDataAvailable

An asynchronous moniker calls this method to provide data to the client as it becomes available during asynchronous bind operations.

```
HRESULT OnDataAvailable(
    DWORD    grfBSCF,          //BSCF enumeration values
    DWORD    dwSize,           //Length of data in bytes available from current bind operation
    FORMATETC *pfmtetc,        //Pointer to FORMATETC structure
    STGMEDIUM *pstgmed         //Pointer to STGMEDIUM structure
  );
```

Parameters

grfBSCF

[in] **DWORD v**alue taken from the **BSCF** enumeration indicating the kind of data available.

dwSize

[in] Size in bytes of total data available from the current bind operation.

pfmtetc

[in] Pointer to the **FORMATETC** structure that indicates the format of the available data. This parameter is used when the bind operation is a result of **IMoniker::BindToStorage**. If there is no format associated with the available data, *pfmtetc* may contain CF_NULL.

pstgmed

[in] Pointer to the **STGMEDIUM** structure that holds the actual data that is now available. This parameter is used when called as a result of **IMoniker::BindToStorage**. If the client 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** function to release the storage. Note that *pstgmed->pUnkForRelease* may be NULL indicating that the storage medium cannot be kept alive. For example, this will be the case when using URL monikers to download data that is not being cached.

Return Values

S_OK

The operation was successful.

E_INVALIDARG

One or more parameters are invalid.

Remarks

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 can be asynchronous or blocking, and the bind operation can follow a data pull model or a data push 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 data 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.

See Also

BINDF, BSCF, IBindStatusCallback::GetBindInfo, IBindStatusCallback::OnDataAvailable

IBindStatusCallback::OnLowResource

An asynchronous moniker calls this method when it detects low resources.

HRESULT OnLowResource(
  DWORD *dwReserved* //Reserved for future use; must be zero
);

Parameters

dwReserved
[in] Reserved for future use; must be zero.

Return Values

S_OK
The operation was successful.

Remarks

The client should free any resource it no longer needs when receiving this notification.

Notes to Implementers

A client can return E_UNIMPL or S_OK if it is not interested in receiving this notification.

See Also

IBindStatusCallback::OnObjectAvailable

An asynchronous moniker calls this method in response to an **IMoniker::BindToObject** bind operation to pass the requested object interface pointer to the client.

HRESULT OnObjectAvailable(
  REFIID *riid*, //Interface identifier of the requested interface
  IUnknown ***punk** //Pointer to the object requested in IMoniker::BindToObject
);

Parameters

riid
  [in] Interface identifier of the requested interface.

punk
  [in] Pointer to the **IUnknown** interface on the object requested in the call to **IMoniker::BindToObject**. The client should call **AddRef** on this pointer to maintain a reference to the object.

Return Values

S_OK
  The operation was successful.

E_INVALIDARG
  One or more parameters are invalid.

Remarks

This method is never called for **IMoniker::BindToStorage** operations.

See Also

IBindStatusCallback::OnDataAvailable

IBindStatusCallback::OnProgress

An asynchronous moniker calls this method to indicate the current progress of the bind operation.

```
HRESULT OnProgress(
    ULONG    ulProgress,        //Progress indicator for bind operation
    ULONG    ulProgressMax,     //Expected maximum value of ulProgress parameter
    ULONG    ulStatusCode,      //BINDSTATUS value indicating bind progress
    LPCWSTR  szStatusText       //Displayable information indicating bind progress
  );
```

Parameters

ulProgress
[in] **ULONG** value indicating the current progress of the bind operation relative to the expected maximum indicated in the *ulProgressMax* parameter.

ulProgressMax
[in] **ULONG** value indicating the expected maximum value of the *ulProgress* parameter for the duration of calls to **IBindStatusCallback::OnProgress** for this bind operation. Note that this value may change across calls to this method.

ulStatusCode
[in] **ULONG**value that provides additional information regarding the progress of the bind operation. Valid values are taken from the **BINDSTATUS** enumeration.

szStatusText
[in] Textual information indicating the current progress of the bind operation. The text reflects the **BINDSTATUS** value of the *ulStatusCode* parameter and is appropriate for display in the user interface of the client.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One or more parameters are invalid.

Remarks

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

The client can use the progress notification to provide progress information to the end user from the *ulProgress*, *ulProgressMax*, and *szStatusText* parameters or to make programmatic decisions based on the *ulStatusCode* parameter.

Notes to Implementers

A client can return E_UNIMPL or S_OK if it is not interested in receiving this notification.

See Also

BINDSTATUS

IBindStatusCallback::OnStartBinding

Notifies the client about the callback methods it is registered for receiving. This notification is a response to the flags the client requested in RegisterBindStatusCallback.

```
HRESULT OnStartBinding(
    DWORD  grfBSCOption,     //BSCO_OPTION flags
    IBinding *pbinding       //Pointer to the IBinding interface for the current bind operation
  );
```

Parameters

grfBSCOption

[in] Flags from the BSCO_OPTION enumeration that specify which callback notifications the client is registered for receiving.

pbinding

[in] Pointer to the **IBinding** interface of the current bind operation. May not be NULL. The client should call **AddRef** on this pointer to keep a reference to the binding object.

Return Values

S_OK

The operation was successful.

E_INVALIDARG

The *pbinding* parameter is invalid.

Remarks

Asynchronous monikers typically call this method while initiating the bind operation in its implementation of **IMoniker::BindToStorage** or **IMoniker::BindToObject**.

In the call to this method, the moniker also provides the client with a pointer to the **IBinding** interface for the binding object associated with the current bind operation. The client can use the **IBinding** interface to control the progress of the bind operation.

To keep a reference to the binding object, the client must store the pointer to the **IBinding** interface and call **AddRef** on it. When the client no longer needs the reference, it must call **Release** on it. Note that calling **Release** does not cancel the bind operation, it simply frees the reference to the **IBinding** interface sent in the callback.

Notes to Implementers

A client can return E_UNIMPL or S_OK if it is not interested in receiving this notification.

See Also

BSCO_OPTION, IBinding, IBindStatusCallback::OnStopBinding, RegisterBindStatusCallback

IBindStatusCallback::OnStopBinding

An asynchronous moniker calls this method to indicate the end of the bind operation.

```
HRESULT OnStopBinding(
    HRESULT   hrStatus,          //Bind operation status code
    LPCWSTR   szStatusText       //Status text
  );
```

Parameters

hrStatus

[in] Status code which would have been returned from the method that initiated the bind operation: either **IMoniker::BindToObject** or **IMoniker::BindToStorage**.

szStatusText

[in] Status text. In case of error, this text describes the error. In case of success, the text displays the friendly name of the bound data location. In the current implementation of URL monikers, this string is empty.

Return Values

S_OK

The operation was successful.

Remarks

This method is always called, whether the bind operation succeeded, failed, or was aborted by a client. At this point the moniker client can use **IBinding::GetBindResult** to query for protocol-specific information about the outcome of the bind operation. Once this method has completed, the moniker client must call **Release** on the **IBinding** pointer it received in **IBindStatusCallback::OnStartBinding**.

Notes to Implementers

A client can return E_UNIMPL or S_OK if it is not interested in receiving this notification.

See Also

IBinding, IBindStatusCallback::OnStartBinding

IBindStatusCallback::QueryInterface

An asynchronous moniker calls this method to query the client for additional services necessary to complete the bind operation.

HRESULT QueryInterface(
  REFIID *riid*, //Interface identifier for the requested service
  **void ****ppvObject* //Pointer to interface returned by the client
);

Parameters

riid
  [in] Interface identifier for the requested service.

szStatusText
  [out] Indirect pointer to the requested interface returned by the client.

Return Values

S_OK
  The operation was successful. The moniker uses the requested interface to communicate further information about the bind operation.

E_NOINTERFACE
  The client does not support the requested interface. Note that if none of the callback interfaces registered for the bind operation can support the requested interface, the bind operation performs a default action.

E_OUTOFMEMORY
  The operation ran out of memory and did not complete successfully.

E_INVALIDARG
  One or more parameters are not valid.

Remarks

This method provides extensibility to the **IBindStatusCallback** interface by allowing the moniker to query its client for additional callback interfaces to notify the client or query the client for further information.

The client of an asynchronous moniker can provide additional callback interfaces through the **IServiceProvider** interface. After the moniker calls **IBindStatusCallback::QueryInterface** to directly query for an extension interface, the moniker can then query for the **IServiceProvider** interface and can call **IServiceProvider::QueryService** to query for the desired extension interface. Because **IServiceProvider::QueryService** is not restricted by COM identity rules that apply to the **IBindStatusCallback::QueryInterface** method, the **IServiceProvider::QueryService** method allows moniker clients to delegate extension services to other objects. Note that if a client is delegating an **IServiceProvider::QueryService** request to a different callback object, the client should first delegate an interface acquired through the second object's **IBindStatusCallback::QueryInterface** before querying the second object for **IServiceProvider**.

See Also

IBinding

ICatInformation

The **ICatInformation** interface provides methods for obtaining information about the categories implemented or required by a certain class, as well as information about the categories registered on a given machine.

When to Implement

There is no need to implement this interface. The Component Category Manager, a system-provided COM object that can be instantiated by using **CoCreateInstance**, implements **ICatInformation**.

When to Use

Call the methods of **ICatInformation** to obtain a listing of available categories, enumerate classes that belong to a particular category, and determine if a class belongs to a specific category.

Methods in Vtable Order

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

ICatInformation Methods	Description
EnumCategories	Returns an enumerator for the component categories registered on the system.
GetCategoryDesc	Retrieves the localized description string for a specific category ID.
EnumClassesOfCategories	Returns an enumerator over the classes that implement/require a certain set of categories.
IsClassOfCategories	Determines if a class implements/requires the specified categories.
EnumImplCategoriesOfClass	Returns an enumerator over the CATIDs implemented by the specified class.
EnumReqCategoriesOfClass	Returns an enumerator over the CATIDs required by the specified class.

See Also

ICatRegister

ICatInformation::EnumCategories

Returns an enumerator for the component categories registered on the system.

```
HRESULT EnumCategories(
    LCID lcid,                          //Requested locale for returned szDescription of enumerated CATEGOR
    REFCLSID rclsid,                    //The CLSID to be queried
    IEnumCATEGORYINFO** ppenumCatInfo   //Location in which to return an IEnumCATEGORYINFO interface
  );
```

Parameters

lcid

[in] Identifies the requested locale for any return *szDescription* of the enumerated CATEGORYINFOs. Typically, the caller specifies **GetUserDefaultLCID** for this parameter.

rclsid

[in] Specifies the CLSID to be queried. If *rclsid* is CLSID_NULL, the enumerator returned will enumerate all component categories.

ppenumCatInfo

[out] Specifies the location to return an **IEnumCATEGORYINFO** interface. This can be used to enumerate the CATIDs and localized description strings of the component categories registered with the system.

Return Values

S_OK

The function was successful.

E_INVALIDARG

One or more arguments are incorrect.

E_OUTOFMEMORY

Insufficient memory to create and return an enumerator object.

See Also

ICatInformation::EnumClassesOfCategories, ICatInformation::EnumImplCategoriesOfClass, ICatInformation::EnumReqCategoriesOfClass, ICatInformation::GetCategoryDesc, ICatInformation::IsClassOfCategories

ICatInformation::GetCategoryDesc

Retrieves the localized description string for a specific category ID.

```
HRESULT GetCategoryDesc(
    REFCATID rcatid,        //Category for which the string is to be returned
    LCID lcid,              //Locale in which the resulting string is returned
    PWCHAR* ppszDesc        //Pointer to the string pointer that contains the description
  );
```

Parameters

rcatid
[in] Identifies the category for which the description string is to be returned.

lcid
[in] Specifies the locale in which the resulting string is returned.

ppszDesc
[out] A pointer to the string pointer that contains the description. This must be released by the caller using **CoMemTaskFree**.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

E_OUTOFMEMORY
Insufficient memory to create and return an enumerator object.

CAT_E_CATIDNOEXIST
The category ID *rcatid* is not registered.

CAT_E_NODESCRIPTION
There is no description string for *rcatid* with the specified locale.

See Also

ICatInformation::EnumCategories, ICatInformation::EnumClassesOfCategories, ICatInformation::EnumImplCategoriesOfClass, ICatInformation::EnumReqCategoriesOfClass, ICatInformation::IsClassOfCategories

ICatInformation::EnumClassesOfCategories

Returns an enumerator over the classes that implement one or more of *rgcatidImpl*. If a class requires a category not listed in *rgcatidReq*, it is not included in the enumeration.

```
HRESULT EnumClassesOfCategories(
    ULONG cImplemented,          //Number of category IDs in the rgcatidImpl array
    CATID rgcatidImpl,           //Array of category identifiers
    ULONG cRequired,             //Number of category IDs in the rgcatidReq array
    CATID rgcatidReq,            //Array of category identifiers
    IEnumCLSID** ppenumCLSID     //Location in which to return an IEnumCLSID interface
  );
```

Parameters

cImplemented
[in] The number of category IDs in the *rgcatidImpl* array. This value cannot be zero.

rgcatidImpl
[in] An array of category identifiers.

cRequired
[in] The number of category IDs in the *rgcatidReq* array. This value can be zero.

rgcatidReq
[in] An array of category identifiers.

ppenumCLSID
[out] The location in which to return an **IEnumCLSID** interface that can be used to enumerate the CLSIDs of the classes that implement category *rcatid*.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

E_OUTOFMEMORY
Insufficient memory to create and return an enumerator object.

See Also

ICatInformation::EnumCategories, ICatInformation::EnumImplCategoriesOfClass, ICatInformation::EnumReqCategoriesOfClass, ICatInformation::GetCategoryDesc, ICatInformation::IsClassOfCategories

ICatInformation::IsClassOfCategories

Determines if a class implements one or more categories. If the class requires a category not listed in *rgcatidReq*, it is not included in the enumeration.

```
HRESULT IsClassOfCategories(
    REFCLSID rclsid,          //Class ID of the class to query
    ULONG cImplemented,       //Number of category IDs in the rgcatidImpl array
    CATID rgcatidImpl,        //Array of category identifiers
    ULONG cRequired,          //Number of category IDs in the rgcatidReq array
    CATID rgcatidReq          //Array of category identifiers
  );
```

Parameters

rclsid
[in] The class ID of the relevent class to query.

cImplemented
[in] The number of category IDs in the *rgcatidImpl* array. This value cannot be zero.

rgcatidImpl
[in] An array of category identifiers.

cRequired
[in] The number of category IDs in the *rgcatidReq* array. This value can be zero.

rgcatidReq
[in] An array of category identifiers.

Return Values

S_OK
rclsid is of category *rcatid*.

S_FALSE
rclsid is not of category *rcatid*.

See Also

ICatInformation::EnumCategories, ICatInformation::EnumClassesOfCategories, ICatInformation::EnumImplCategoriesOfClass, ICatInformation::EnumReqCategoriesOfClass, ICatInformation::GetCategoryDesc

ICatInformation::EnumImplCategoriesOfClass

Returns an enumerator for the CATIDs implemented by the specified class.

HRESULT EnumImplCategoriesOfClass(
  REFCLSID *rclsid*, //Class ID
  IEnumCATID** *ppenumCATD* //Location in which to return an **IEnumCATID** interface
);

Parameters

rclsid
[in] Specifies the class ID.

ppenumCATD
[out] Specifies the location to return an **IEnumCATID** interface. This can be used to enumerate the CATIDs that are implemented by *rclsid*.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

E_OUTOFMEMORY
Insufficient memory to create and return an enumerator object.

See Also

ICatInformation::EnumCategories, ICatInformation::EnumClassesOfCategories, ICatInformation::EnumReqCategoriesOfClass, ICatInformation::GetCategoryDesc, ICatInformation::IsClassOfCategories

ICatInformation::EnumReqCategoriesOfClass

Returns an enumerator for the CATIDs required by the specified class.

```
HRESULT EnumReqCategoriesOfClass(
    REFCLSID rclsid,                //Class ID
    IEnumCATID** ppenumCATD         //Location in which to return an IEnumCATID interface
  );
```

Parameters

rclsid
[in] Specifies the class ID.

ppenumCATD
[out] Specifies the location to return an **IEnumCATID** interface. This can be used to enumerate the CATIDs that are required by *rclsid*.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

E_OUTOFMEMORY
Insufficient memory to create and return an enumerator object.

See Also

ICatInformation::EnumCategories, ICatInformation::EnumClassesOfCategories, ICatInformation::EnumImplCategoriesOfClass, ICatInformation::GetCategoryDesc, ICatInformation::IsClassOfCategories

ICatRegister

The **ICatRegister** interface provides methods for registering and unregistering component category information in the Registry. This includes both the human-readable names of categories and the categories implemented/required by a given component or class.

When to Implement

There is no need to implement this interface. The Component Category Manager, a system-provided COM object that can be instantiated by using **CoCreateInstance**, implements **ICatRegister**.

When to Use

The owner of a category uses this interface to register or unregister the human-readable names. The owner of a component uses this interface to add or remove categories implemented or required by this component.

Methods in Vtable Order

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

ICatRegister Methods	Description
RegisterCategories	Registers one or more component categories.
UnRegisterCategories	Removes the registration of one or more component categories.
RegisterClassImplCategories	Registers the class as implementing one or more component categories.
UnRegisterClassImplCategories	Removes one or more implemented category identifiers from a class.
RegisterClassReqCategories	Registers the class as requiring one or more component categories.
UnRegisterClassReqCategories	Removes one or more required category identifiers from a class.

Component Categories

As the number of available components grows, it becomes increasingly difficult to manage these components. Many components offer similar functionality, both in terms of the interfaces they expose as well as the tasks they perform.

In many situations, it is necessary to enumerate the components that can be used in a certain context. Examples of this are the Insert Object dialog box used in OLE Compound Documents and the Insert Control dialog box used in OLE Controls. These dialog boxes list all components that fulfill (or claim to fulfill) the interface contracts for compound documents or controls. These existing categories (OLE Document, OLE Control) do not imply an exact interface signature. OLE Documents have to expose a certain set of core interfaces (for example, **IOleObject** or **IPersistStorage**), but can also expose additional interfaces such as **IOleLink**.

In the past, components have been tagged by adding a human-readable name (“Insertable”, “Control”, and so on) as a sub-key to the HKEY_CLASSES_ROOT\CLSID\{...} key corresponding to the component. This works well for a central definition of categories, but runs the danger of name collisions when many independent parties define new categories. As in other areas of COM, the solution to providing an extensible name space lies in the use of globally unique identifiers (GUIDs). Instead of using a human-readable name, a unique number (CATID) is assigned to each category.

Another limitation with the existing categorization is that it is limited to expressing the capabilities of the component itself. Many components require certain capabilities from the containers. When such a component is inserted into a container, the insertion can fail or behave unexpectedly, even though the component fulfills the contracts implied by one of its categories. To enumerate the components that can be used successfully in certain situations, both the capabilities of the component and the container must be considered.

Because of these considerations, the following changes were made to the existing categorization:

- Categories are indicated by using CATIDs that are globally unique identifiers.
- Under the Components sub-key of the HKEY_CLASSES_ROOT\CLSID Registry key, two separate sub-keys, “Implemented Categories” and “Required Categories”, were developed. These sub-keys contain the lists of CATIDs that are provided by the component or that the container of the component must provide.

To ease managing the component categories, categories are listed in a central place in the Registry: HKEY_CLASSES_ROOT\Component Categories. This key lists the installed categories with both their CATID and localized, human-readable names.

Categorizing by Component Capabilities

Component categories can be used to display a subset of all of the installed components. Each component category is identified by a GUID, referred to as a Category ID (CATID). Each CATID has a list of locale-tagged, human-readable names associated with it. A listing of the CATIDs and the human-readable names is stored in a well-known location in the Registry.

For example, all components that implement the functionality for OLE document embedding can be classified within a component category. In the past, these objects would have been identified by the "Insertable" key in the Registry. To use component categories instead, the following information would be added to the Registry:

```
HKEY_CLASSES_ROOT\Component Categories\{40FC6ED3-2438-11cf-
A3DB-080036F12502}
                    = (default) - ""
                    = 409 - "OLE Document Embeddings"
```

Each class that implements the functionality corresponding to a component category lists the category ID for that category within the CLSID key in the Registry. Because a single component can support a wide range of functionality, components can belong to multiple component categories. For example, a particular OLE control might support all of the functionality required to participate as OLE document embedding, Visual Basic data binding, and internet functionality. Such a control would have the following information stored in its CLSID key in the Registry:

```
;The CLSID for "My Super OLE Control" is {12345678-ABCD-4321-0101-
00000000000C}HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-
00000000000C}\Implemented Categories
;The CATID for "Insertable" is {40FC6ED3-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}Implemented
Categories\{40FC6ED3-2438-11cf-A3DB-080036F12502}
;The CATID for "Control" is {40FC6ED4-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}Implemented
Categories\{40FC6ED4-2438-11cf-A3DB-080036F12502}
;The CATID for an internet aware control is {...CATID_InternetAware...}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}Implemented
Categories\{...CATID_InternetAware...}
```

With this information, a container can enumerate the controls installed on a system and display only those controls that support the functionality required by the container. Component categories provides a way to categorize components by the implemented functionality of the component.

Categorizing by Container Capabilities

The previous section showed how to use component categories to allow a container to enumerate the controls installed on a system, and to display only the controls that support a functionality required by the container. However, components often require certain functionality from the container and will not work with a container that does not provide the support. The user interface should filter out components that require functionality the container does not support. To accomplish this, component categories can also categorize components by the required container functionality.

An example of components that require functionality from the container and do not work in containers that do not support that functionality are simple frame OLE controls. Categorizing by container capabilities is accomplished by an additional Registry key within the component's CLSID key:

```
;The CLSID for "Simple Frame Control" is {123456FF-ABCD-4321-0101-
00000000000C}HKEY_CASSES_ROOT\CLSID\{12346FF-ABCD-4321-0101-00000000000C}
\Implemented Categories
;The CATID for "Control" is {40FC6ED4-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}Implemented
Categories\{40FC6ED4-2438-11cf-A3DB-080036F12502}
;The CATID for simple frame controls is {...CATID_SimpleFrameControl...}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}Implemented
Categories\{...CATID_SimpleFrameControl...}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}Required
Categories\{...CATID_SimpleFrameControl...}
```

As shown in this example, a component can belong to component categories that indicate supported functionality and component categories that indicate required functionality.

In the following example, the button control is a generic OLE control that supports no additional functionality. It will work in any OLE control container.

```
HKEY_CLASSES_ROOT\CLSID\{...CLSID_Button...}\Implemented Categories
HKEY_CLASSES_ROOT\CLSID\{...CLSID_Button...}\Implemented Categories\
{...CATID_Control...}
```

Compare the example above with the next example in which the MyDBControl can use Visual Basic data binding if the container supports it. However, it has been defined so that it will work in containers that do not support Visual Basic data binding (perhaps by a different database API):

```
HKEY_CLASSES_ROOT\CLSID\{...CLSID_MyDBControl...}\Implemented Categories
HKEY_CLASSES_ROOT\CLSID\{...CLSID_MyDBControl...}\Implemented Categories\
{...CATID_Control...}
HKEY_CLASSES_ROOT\CLSID\{...CLSID_MyDBControl...}\Implemented Categories\
{...CATID_VBDatabound...}
```

The GroupBox control is a simple frame control. It relies on the container implementing the ISimpleFrameSite interface, and will only work correctly in such containers:

```
HKEY_CLASSES_ROOT\CLSID\{...CLSID_GroupBox...}\Implemented Categories
HKEY_CLASSES_ROOT\CLSID\{...CLSID_GroupBox...}\Implemented Categories\
{...CATID_Control...}
HKEY_CLASSES_ROOT\CLSID\{...CLSID_GroupBox...}\Implemented Categories\
{...CATID_SimpleFrame...}
HKEY_CLASSES_ROOT\CLSID\{...CLSID_GroupBox...}\Required Categories\
{...CATID_SimpleFrame...}
```

A container that supports Visual Basic data bound controls, but does not support simple frame controls would specify CATID_Control and CATID_VBDatabound to the insert control user interface. The list of controls displayed to the user would contain the CLSID_Button and CLSID_MyDBControl. CLSID_GroupBox would not be displayed.

Default Classes and Associations

For certain categories, a single class can be associated as the default class. The default class is selected whenever that particular category of object is required. While this may not be useful for all component categories, establishing a default class can be helpful when a particular class must be loaded from a list of possible classes without user intervention. Administrators define which class can be used by manipulating the registry.

To associate a default class with a category, introduce a CLSID key with the same CLSID as the CATID of the component category chosen as the default. Then, add a TreatAs key to this key, using the value for the CLSID of the default class for the category. To use the default class for a component category, use **CoCreateInstance** or **CoGetClassObject**, specifying the CATID for the CLSID parameter. This automatically redirects to the CLSID established as the default for this category. The Registry entry is as follows:

```
HKEY_CLASSES_ROOT\CLSID\{...catid...}\TreatAs = {...default clsid...}
```

During installation, a component can check for the existence of any default class keys for its categories and present the user with options for overriding the current settings.

Defining Component Categories

The author of a component category definition creates a unique GUID (the CATID) which is published along with the definition. Other parties know the definition of this type and can make use of its supported classes accordingly. Like the method signature of an interface, a category's semantics should not be modified after being installed. It is better to maintain backward compatibility of the category by introducing a new category identifier with revised semantics.

Because interface identifiers (IID) and component category identifiers (CATID) exist in different name spaces, it seems as though it would be possible to use the same GUID for both an IID and a CATID. However, since IIDs are often used for the CLSID of the interface's proxy/stub server, there is the potential for conflict. Therefore, do not use the same GUID for an IID and CATID.

Associating Icons with a Category

Building a user interface that allows the user to select component categories within a category requires the ability to display a meaningful image for a particular category. To associate an icon with a component category, create a key for the category's CATID and populate that key with a DefaultIcon sub-key. The Registry entry is as follows:

```
HKEY_CLASSES_ROOT\CLSID\{...catid...}\DefaultIcon = "c:
\hello\icons.dll,1"
```

The filename referenced by the DefaultIcon key can be either a .EXE or .DLL file (resource-only .DLL).

To associate small 16x16 “toolbox bitmap” with a component category, create a key in HKEY_CLASSES_ROOT\CLSID for the category’s CATID and populate that key with a TolboxBitmap32 sub-key. For example:

```
HKEY_CLASSES_ROOT\CLSID\{...catid...}\ToolboxBitmap32 = "c:
\goodbye\mycomponent.dll,42"
```

The filename referenced by the ToolboxBitmap32 key can also be a .EXE or .DLL file (resource-only .DLL).

See Also

EnumCategories, EnumClassesOfCategories, EnumImplCategoriesOfClass, EnumReqCategoriesOfClass, GetCategoryDesc, ICatInformation, IsClassOfCategories, RegisterCategories, RegisterClassImplCategories, RegisterClassReqCategories, UnRegisterCategories, UnRegisterClassImplCategories, UnRegisterClassReqCategories

ICatRegister::RegisterCategories

Registers one or more component categories. Each component category consists of a CATID and a list of locale-dependent description strings.

HRESULT RegisterCategories(
  ULONG *cCategories***,** //Number of component categories
  CATEGORYINFO* *rgCategoryInfo* //Array of *cCategories* CATEGORYINFO structures
);

Parameters

cCategories
  [in] The number of component categories to register.

rgCategoryInfo
  [in] The array of *cCategories* CATEGORYINFO structures. By providing the same CATID for multiple CATEGORYINFO structures, multiple locales can be registered for the same component category.

Return Values

S_OK
  The function was successful.

E_INVALIDARG
  One or more arguments are incorrect.

Remarks

This function can only be called by the owner of a category, usually as part of the installation or de-installation of the operating system or application.

See Also

ICatRegister::RegisterClassImplCategories, ICatRegister::RegisterClassReqCategories, ICatRegister::UnRegisterCategories, ICatRegister::UnRegisterClassImplCategories, ICatRegister::UnRegisterClassReqCategories

ICatRegister::UnRegisterCategories

Removes the registration of one or more component categories. Each component category consists of a CATID and a list of locale-dependent description strings.

```
HRESULT UnRegisterCategories(
    ULONG cCategories,      //Number of cCategories CATIDs to be removed
    REFCATID rgcatid        //Array of cCategories CATIDs
  );
```

Parameters

cCategories
[in] The number of *cCategories* CATIDs to be removed.

rgcatid
[in] Identifies the categories to be removed.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

This function will be successful even if one or more of the category IDs specified are not registered. This function can only be called by the owner of a category, usually as part of the installation or de-installation of the operating system or application.

Note This method does not remove the component category tags from individual classes. To do this, use the **ICatRegister::UnRegisterClassCategories** method.

See Also

ICatRegister::RegisterCategories, ICatRegister::RegisterClassImplCategories, ICatRegister::RegisterClassReqCategories, ICatRegister::UnRegisterClassImplCategories, ICatRegister::UnRegisterClassReqCategories

ICatRegister::RegisterClassImplCategories

Registers the class as implementing one or more component categories.

```
HRESULT RegisterClassImplCategories(
    REFCLSID rclsid,          //Class ID of the relevent class
    ULONG cCategories,        //Number of category CATIDs
    CATID* rgcatid            //Array of cCategories CATID
  );
```

Parameters

rclsid
[in] The class ID of the relevent class for which category information will be set.

cCategories
[in] The number of category CATIDs to associate as category identifiers for the class.

rgcatid
[in] The array of *cCategories* CATID to associate as category identifiers for the class.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

In case of an error, this function does not ensure that the Registry is restored to the state prior to the call. This function can only be called by the owner of a class, usually as part of the installation of the component.

See Also

ICatRegister::RegisterCategories, ICatRegister::RegisterClassReqCategories, ICatRegister::UnRegisterCategories, ICatRegister::UnRegisterClassImplCategories, ICatRegister::UnRegisterClassReqCategories

ICatRegister::UnRegisterClassImplCategories

Removes one or more implemented category identifiers from a class.

HRESULT UnRegisterClassImplCategories(
  REFCLSID *rclsid*, //Class ID of the relevent class
  ULONG *cCategories*, //Number of category CATIDs
  CATID* *rgcatid* //Array of *cCategories* CATID
);

Parameters

rclsid
[in] The class ID of the relevant class to be manipulated.

cCategories
[in] The number of category CATIDs to remove.

rgcatid
[in] The array of *cCategories* CATID that are to be removed. Only the category IDs specified in this array are removed.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

In case of an error, this function does not ensure that the Registry is restored to the state prior to the call. The call will be successful even if one or more of the category IDs specified are not registered for the class. This function can only be called by the owner of a class, usually as part of the de-installation of the component.

See Also

ICatRegister::RegisterCategories, ICatRegister::RegisterClassImplCategories, ICatRegister::RegisterClassReqCategories, ICatRegister::UnRegisterCategories, ICatRegister::UnRegisterClassReqCategories

ICatRegister::RegisterClassReqCategories

Registers the class as requiring one or more component categories.

```
HRESULT RegisterClassReqCategories(
    REFCLSID rclsid,          //Class ID of the relevent class
    ULONG cCategories,        //Number of category CATIDs
    CATID* rgcatid            //Array of cCategories CATID
  );
```

Parameters

rclsid
[in] The class ID of the relevent class for which category information will be set.

cCategories
[in] The number of category CATIDs to associate as category identifiers for the class.

rgcatid
[in] The array of *cCategories* CATID to associate as category identifiers for the class.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

In case of an error, this function does not ensure that the Registry is restored to the state prior to the call. This function can only be called by the owner of a class, usually as part of the installation of the component.

See Also

ICatRegister::RegisterCategories, ICatRegister::RegisterClassImplCategories, ICatRegister::UnRegisterCategories, ICatRegister::UnRegisterClassImplCategories, ICatRegister::UnRegisterClassReqCategories

ICatRegister::UnRegisterClassReqCategories

Removes one or more required category identifiers from a class.

```
HRESULT UnRegisterClassReqCategories(
    REFCLSID rclsid,          //Class ID of the relevent class
    ULONG cCategories,        //Number of category CATIDs
    CATID* rgcatid*           //Array of cCategories CATID
  );
```

Parameters

rclsid
[in] The class ID of the relevent class to be manipulated.

cCategories
[in] The number of category CATIDs to remove.

rgcatid
[in] The array of *cCategories* CATID that are to be removed. Only the category IDs specified in this array are removed.

Return Values

S_OK
The function was successful.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

In case of an error, this function does not ensure that the Registry is restored to the state prior to the call. The call will be successful even if one or more of the category IDs specified are not registered for the class.

See Also

ICatRegister::RegisterCategories, ICatRegister::RegisterClassImplCategories, ICatRegister::UnRegisterCategories, ICatRegister::UnRegisterClassImplCategories, ICatRegister::RegisterClassReqCategories

ICodeInstall

The **ICodeInstall** interface provides additional information in order to complete the negotiation for a component download operation. Such services are requested using **IBindStatusCallback::QueryInterface**.

When to Implement

Implement **ICodeInstall** to receive further communications beyond those provided with the IBindStatusCallback methods.

When to Use

Typically, the **ICodeInstall** interface is a callback interface used by system components to communicate with clients. Specifically, the Internet Component Download service (**CoGetClassObjectFromURL**) calls the methods of **ICodeInstall** to display a user interface (UI) message for verification of downloaded code. Methods of this interface are also used to provide an opportunity to resolve a problem, either by displaying a UI message or ending the code installation process.

Methods in Vtable Order

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

ICodeInstall Methods	Description
GetWindow	Displays a UI message to verify downloaded code.
OnCodeInstallProblem	Provides an opportunity to resolve a problem with the code installation.

See Also

IBindStatusCallback, ICodeInstall::GetWindow, ICodeInstall::OnCodeInstallProblem

ICodeInstall::GetWindow

Displays a user interface (UI) message to verify downloaded code. When a client is called with this method, the message queue can be cleared before allowing the UI to be displayed.

HRESULT GetWindow(
  HWND * *phwnd* //Handle for the parent window that displays code verification
);

Parameters

phwnd
  [out] Handle provided by the client for the parent window that displays code verification. If this parameter is NULL, the desktop window is used. If the value is INVALID_HANDLE_VALUE or if the return value is S_FALSE, no code verification UI will be displayed and certain necessary components may not be installed.

Return Values

S_OK
  The function was successful.

S_FALSE
  No window is available.

E_INVALIDARG
  The argument is incorrect.

See Also

ICodeInstall::OnCodeInstallProblem

ICodeInstall::OnCodeInstallProblem

Provides an opportunity to resolve a problem with the download operation. Calling this method provides information about a problem with the code installation by displaying a UI message. If the problem cannot be understood, E_ABORT should be returned by default to stop the code installation process. Otherwise, a return of S_OK would imply retrying the operation.

```
HRESULT OnCodeInstallProblem(
    ULONG ulStatusCode,         //Describes the problem that has occurred
    LPCWSTR szDestination,      //Name of the existing file causing the problem
    LPCWSTR szSource,           //Name of the new file to replace the existing file
    DWORD dwReserved            //Reserved
  );
```

Parameters

ulStatusCode
[in] The status code describing the problem that has occurred.

szDestination
[in] The name of the existing file causing the problem. This can be the name of an existing file that needs to be overwritten, the name of a directory causing access problems, or the name of a drive that is full.

szSource
[in] If applicable, the name of the new file to replace the existing file.

dwReserved
[in] Reserved for future use.

Return Values

S_OK
Continue the installation process. If there was an access denied or full disk problem, retry the installation. If there was an existing file (newer or older version), overwrite it.

S_FALSE
Omit this particular file, but continue with the rest of the code installation process. This is the typical response for the CIP_NEWER_VERSION_EXISTS status code.

E_ABORT
Stop the code installation process.

E_INVALIDARG
One or more arguments are incorrect.

Remarks

The *ulStatusCode* parameter contains one of the following values:

CIP_DRIVE_FULL
The drive specified in *szDestination* is full.

CIP_ACCESS_DENIED
Access to the file specified in *szDestination* is denied.

CIP_OLDER_VERSION_EXISTS
An existing file (older version) specified in *szDestination* needs to be overwritten by the file specified in *szSource*.

CIP_NEWER_VERSION_EXISTS

A file exists (specified in *szDestination*) that is a newer version of a file to be installed (specified in *szSource*).

CIP_NAME_CONFLICT
A file exists (specified in *szDestination*) that has a naming conflict with a file to be installed (specified in *szSource*). The existing file is neither a new or an older version of the file; rather, they are mismatched but have the same filename.

CIP_TRUST_VERIFICATION_COMPONENT_MISSING
The code installation process cannot find the necessary component (**WinVerifyTrust**) for verifying trust in downloaded code. *szSource* specifies the name of the file that cannot be certified. The client should display a UI message asking the user whether or not to install the untrusted code, and should then return E_ABORT to stop the downloading, S_OK to continue the downloading, or S_FALSE to omit this file but continue.

Note Use caution when returning S_FALSE to omit the file but continue the downloading operation.

See Also

ICodeInstall::GetWindow

IContinueCallback

This interface is a generic callback mechanism for interruptible processes that should periodically ask an object with this interface whether to continue the process.

The *FContinue* function is a generic continuation request. *FContinuePrinting* carries extra information pertaining to a printing process and is used in the contect of *IPrint.*

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces
AddRef	Increments reference count
Release	Decrements reference count

IContinueCallback Methods	Description
FContinue	Answers whether an operation should continue
FContinuePrinting	Answers whether a printing operation should continue

IContinueCallback::FContinue

Answer as to whether a given generic operation should continue.

HRESULT FContinue(void);

Return Values

S_OK
: Continue the operation.

S_FALSE
: Cancel the operation as soon as possible

IContinueCallback::FContinuePrinting

Answer whether a given lengthy printing operation should continue.

```
HRESULT FContinuePrinting(
    LONG cPagesPrinted,          // Total pages printed at time of call
    LONG nCurrentPage,           // Number of page currently being printed
    LPOLESTR pszPrintStatus      // Pointer to status message about print job
  );
```

Parameters

cPagesPrinted

The total number of pages that have been printed at the time the object receives a call to **IContinueCallback::FContinuePrinting**.

nCurrentPage

The page number of the page being printed at the time the object receives a call to **IContinueCallback::FContinuePrinting**.

pszPrintStatus

Pointer to message about the current status of the print job. The object being printed may or may not display this message to the user. May be NULL.

Return Values

S_OK

Continue the printing operation.

S_FALSE

Cancel the print job as soon as possible.

E_UNEXPECTED

An unknown error occurred.

Remarks

Implementations of IPrint call back on this method at periodic intervals during the printing process. The **IPrint** implementation should call back at least after printing each page, so that its client can, if necessary, display useful visual feedback to the user. Your **Iprint** implementation can legally call back multiple times with the same *cPagesPrinted* and *nCurrentPage* values, which is sometimes useful when a page being printed is complex and it is appropriate to give a user an opportunity to cancel mid-page.

See Also

IPrint

IDataPathBrowser

This interface offers extended type-specific browsing capabilities such that a control can invoke a container-provided property-browsing user-interface if desired. Any property page that wishes to invoke a browsing user interface can query for **IDataPathBrowser** on the property page site object that it receives through **IPropertyPage::SetPageSite**. The presence of **IDataPathBrowser** indicates the possibility that the container supplies some type-specific browsing user interface. To invoke the container-provided user interface, the control calls the **IDataPathBrowser::BrowseType** method, passing the GUID identifying the type along with a default path and a buffer in which to store the selected path. If the browser supports the type identified with the GUID, it will display a user interface for that type and return any selection to the control.

Methods in Vtable Order

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

IDataPathBrowser Methods	Description
BrowseType	Invokes a browsing user interface for the specified type.

IDataPathBrowser::BrowseType

Instructs the browser to invoke a browsing user interface for the type identified with *rguidPathType*, if the user interface is available. If the user selects a path to data from within the user interface, the browser should return the string for that path in *pszPath*.

```
HRESULT BrowseType(
   REFGUID   rguidPathType,     // GUID identifying the data type to browse
   LPOLESTR  pszDefaultType,    // Default path string to display in the browsing user interface
   ULONG     cchPath,           // Size of buffer pointed to by pszPath
   LPOLESTR  pszPath,           // Receives the selected string
   HWND      hWnd               // Parent window of the browsing user interface
  );
```

Parameters

rguidPathType
[in] The GUID identifying the data type to browse.

pszDefaultPath
[in] The default path string to display in the browsing user interface.

cchPath
[in] The size of the buffer pointed to by *pszPath*.

pszPath
[out] The buffer to receive the selected string if the browsing is successful.

hWnd
[in] The window that should be the parent of the browsing user interface.

Return Values

S_OK
The browsing user interface was successfully displayed and a new path is returned in *pszPath.*

S_FALSE
The browsing user interface was successfully displayed but the user did not select a path (for example, the user may have pressed Cancel).

E_OUTOFMEMORY
There was not enough memory to display the user interface, but the browsing user interface is available.

E_FAIL
The site does not support browsing for the type specified in *rguidPathType*.

Remarks

E_NOTIMPL is not allowed—without implementing the **BrowseType** method, the **IDataPathBrowser** interface is unnecessary.

IEnumHLITEM

The **IEnumHLITEM** interface is used to enumerate an array of **HLITEM** structures in a hyperlink browse context navigation stack. **IEnumHLITEM** has the same methods as all enumerator interfaces: **Next, Skip, Reset, Clone.** For general information on these methods, refer to **IEnum*XXXX***.

When to Implement

IEnumHLITEM must be implemented by all hyperlink browse context objects to support calls to **IHlinkBrowseContext::EnumNavigationStack**, which supplies a pointer to the enumerator's **IEnumHLITEM** interface. Because there is a default implementation of **IHlinkBrowseContext**, you do not normally need to implement this interface.

When to Use

You do not normally need to call this interface directly. The default implementation of **IHlinkBrowseContext** calls this interface to examine the **HLITEM** structures in the browse context's navigation stack to determine which hyperlink sources have been previously visited.

The prototypes of the methods are as follows:

```
HRESULT Next(
  ULONG celt,              // Number of HLITEM structures returned in rgelt
  HLITEM * rgelt,          // Array to receive enumerated HLITEM structures
  ULONG *pcFetched         // Location for actual number of HLITEM structures
);

HRESULT Skip(
  ULONG cHlitems      // Number of elements to skip
);

HRESULT Reset(void);

HRESULT Clone(
  IEnumHLITEMs **ppEnumHlitems      // Location for newly created enumerator list
);
```

Remarks

IEnumHLITEM::Next

Retrieves the next *celt* **HLITEMs** in the enumeration sequence. If there are fewer than the requested number of elements left in the sequence, it retrieves the remaining elements. The number of elements actually retrieved is returned through *pceltFetched*.

IEnumHLITEM::Skip

Skips the next *celt* **HLITEMs** in the enumeration sequence.

IEnumHLITEM::Reset

Resets the enumeration sequence to the beginning. There is no guarantee that the same set of elements will be enumerated after the **Reset**, because it depends on the implementation doing the enumeration. It can be too expensive for some collections to guarantee this condition or it may not be possible due to concurrent access to the same collection by multiple threads or processes.

IEnumHLITEM::Clone

Creates another enumerator that contains the same state as the current enumerator to iterate over the same list. Using this function, a client can record a particular point in the enumeration sequence, and then return to that point at a later time.

The caller must release this new enumerator separately from the first enumerator.

See Also

HLITEM, **IEnum*XXXX***, IHlinkBrowseContext::EnumNavigationStack

IEnumHLITEM::Next

Retrieves the next celt **HLITEM**s in the enumeration sequence.

```
HRESULT Next(
    ULONG celt,              // Number of elements to retrieve
    HLITEM * rgelt,          // Location to return at most celt
    ULONG * pceltFetched     // Location to return actual number of elements returned in rgelt
  );
```

Parameters

celt

[in] The number of elements to retrieve.

rgelt

[out] Location to return a maximum of *celt* **HLITEM** structures. This may not be NULL.

pceltFetched

[out] Location to return the actual number of elements returned in *rgelt*. May be NULL if the caller is not interested in the actual number of elements returned.

Return Values

S_OK

Indicates all of the elements specified by *celt* have been successfully supplied.

S_FALSE

Indicates fewer than the number of elements specified by *celt* have been successfully supplied. The number actually supplied (including zero) is in *rgelt* and their values reached through *pceltFetched*, if non-NULL.

See Also

TBD

IEnumHLITEM::Skip

Skips the next celt **HLITEM**s in the enumeration sequence.

```
HRESULT Skip(
    ULONG celt      // Number of elements to skip
  );
```

Parameters

celt
[in] The number of elements to skip.

Return Values

S_OK
Indicates all of the elements specified by *celt* have been successfully skipped.

S_FALSE
Indicates fewer than the number of elements specified by *celt,* and possibly zerio elements, have been successfully skipped. This indicates that the enumerator has reached the end of the collection.

See Also

TBD

IEnumHLITEM::Reset

Resets the enumeration sequence to the beginning.

HRESULT Reset(
);

Return Values

S_OK
: Indicates the enumeration sequence has been reset to the beginning.

See Also

TBD

IEnumHLITEM::Clone

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

```
HRESULT Clone(
    IEnumHLITEM ** ppenumhlitem      // Location to return the new enumerator
  );
```

Parameters

ppenumhlitem
[out] Location to return the new enumerator. Must be cleared to NULL on failure.

Return Values

S_OK
Indicates the enumerator has been duplicated successfully.

E_OUTOFMEMORY
Indicates insufficient memory to create the new enumerator.

IEnumOleDocumentViews

The **IEnumOleDocumentViews** interface is used to enumerate the views supported by a document object. **IEnumOleDocumentViews** has the same methods as all enumerator interfaces: **Next**, **Skip**, **Reset**, and **Clone**. For general information on these methods, see **IEnumXXX**.

The resulting enumerator returned by this mtehod implements the interface **IEnumOleDocumentViews** through which a client can access all the individual document view objects supported by the document object itself, where each view implements IOleDocumentView.

When to Implement

Implement **IEnumOleDocumentViews** on enumerator objects associated with document objects that support more than one view of their data.

When to Use

Use the **IEnumOleDocumentViews** to enumerate all the views supported by a document object. The usual procedure is to first call IOleDocument::EnumViews. If the document object supports only one view, it will return a pointer to that view. If the document object supports two or more views, it will return a pointer to **IEnumOleDocumentViews.** Using this pointer, the container can then ask the document object to enumerate the views it supports.

The prototypes of the methods are as follows:

HRESULT Next(ULONG *cViews*, **IOleDocumentView *** *rgpView*, [out] **ULONG *** *pcFetched***)**

HRESULT Skip([in] **ULONG** *cViews*

HRESULT Reset(void)

HRESULT Clone([out] **IEnumOleDocumentViews** ** *ppenum***)**

Remarks

IEnumOleDocumentViews::Next

Enumerates the next *cViews* elements in the enumerator's list, returning them in *rgpView,* along with the actual number of enumerated elements in *pcFetched.* The caller is responsible for calling **IOleDocumentView::Release** through each pointer returned in *rgpView*.

E_NOTIMPL is not allowed as a return value. If an error value is returned, no entries in the *rgpView* array are valid on exit and require no release.

cViews

[in] Specifies the number of **IOleDocumentView** pointers to return in the array pointed to by *rgpView*. This argument must be 1 if *pcFetched* is NULL.

rgpView

[out, max_is(cViews)] A pointer to a caller-allocated **IOleDocumentView** * array of size *cViews* in which to return the enumerated connection points. The caller is responsible for calling **IOleDocumentView::Release** through each pointer enumerated into the array once this method returns successfully. If *cViews* is greater than 1 the caller must also pass a non-NULL pointer passed to *pcFetched* to know how many pointers to release.

pcFetched

[out] A Indirect pointer to the actual number of views enumerated in *rgpView*. This argument can be NULL in which case the *cViews* argument must be 1.

IEnumOleDocumentViews::Skip

Instructs the enumerator to skip the next *cViews* elements in the enumeration such that the next call to **IEnumOleDocumentViews::Next** will not return those elements.

cViews

[in] Specifies the number of elements to skip in the enumeration.

IEnumOleDocumentViews::Reset

Instructs the enumerator to position itself at the beginning of the list of elements. There is no guarantee that the same set of elements will be enumerated on each pass through the list: it depends on the collection being enumerated. It is too expensive for some collections, such as files in a directory, to maintain this condition.

IEnumOleDocumentViews::Clone

Creates another view enumerator with the same state as the current enumerator, which iterates over the same list. This makes it possible to record a point in the enumeration sequence in order to return to that point at a later time.

ppEnum

[out] A Indirect pointer to the **IEnumOleDocumentViews** interface for the newly created enumerator. The caller must release this enumerator separately from the one from which it was cloned.

IErrorLog

The **IErrorLog** interface is an abstraction for an error log that is used to communicate detailed error information between a client and an object. The caller of the single interface method, **AddError**, simply logs an error where the error is an **EXCEPINFO** structure attached to a specific property. The implementor of the interface is responsible for handling the error in whatever way it desires.

IErrorLog is used in the protocol between a client that implements **IPropertyBag** and an object that implements **IPersistPropertyBag**.

Methods in Vtable Order

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

IErrorLog Method	Description
AddError	Logs an error (an **EXCEPINFO** structure) in the error log for a named property.

See Also

IPersistPropertyBag, IPropertyBag

IErrorLog::AddError

Logs an error (an **EXCEPINFO** structure) in the error log for a named property.

HRESULT AddError(
  LPCOLESTR *pszPropName*, // Points to the name of the property involved with the error
  LPEXCEPINFO *pException* // Points to the caller-initialized **EXCEPINFO** structure describing the error
);

Parameters

pszPropName
  [in] The name of the property involved with the error. Cannot be NULL.

pExcepInfo
  [in] The address of the caller-initialized **EXCEPINFO** structure that describes the error to log. Cannot be NULL.

Return Values

S_OK
  The error was logged successfully.

E_FAIL
  There was a problem logging the error.

E_OUTOFMEMORY
  There was not enough memory to log the error.

E_POINTER
  The address in *pszPropName* or *pExceptInfo* is not valid (such as NULL). The caller must supply both.

Remarks

E_NOTIMPL is not a valid return code as the method is the only one in the entire interface.

IHlink

The **IHlink** interface enables a COM object known as a *hyperlink* to completely encapsulate the behavior of navigating to its target location.

Hyperlinks are managed by COM hyperlink container objects, which support the IHlinkSite interface for hyperlinks within the container and the IHlinkTarget interface for hyperlinks referring to documents or document objects within it.

IHlink provides methods for a hyperlink object to navigate to its target, access a friendly name for display purposes, and identify itself to its container and frame.

When to Implement

Typically, you do not need to implement this interface. A standard implementation of a hyperlink component is provided as part of the system, and it is not advisable to implement another version. A document can use the standard hyperlink object to represent hyperlinks within itself, thus encapsulating the capability of navigating, saving, loading, dragging, dropping, cutting, and pasting hyperlinks. The standard hyperlink object implements **IHlink, IPersistStream,** and **IDataObject** interfaces.

When to Use

IHlink interfaces are typically called by an application's hyperlink container as part of a user interface for creating new hyperlinks. Standard hyperlink objects are created via the HlinkCreateFromData, HlinkCreateFromMoniker, HlinkCreateFromString, and **OleLoadFromStream** APIs.

While the hyperlink object provides many useful functions that are needed for general purpose uses, often applications are only interested in navigating to a hyperlink target. For simple navigation needs, there are a number of "helper" APIs such as HlinkSimpleNavigateToString, HlinkSimpleNavigateToMoniker, and HlinkNavigateToStringReference that allow simple hyperlink navigation without any knowledge of other hyperlink interfaces or objects.

Hyperlink navigation involves transitioning from one document/object/application, known as the *hyperlink container*, to another document/object/application, known as the *hyperlink target*. Usually, both remain running, but the hyperlink target visually replaces the hyperlink container (from which the navigation originated). There are three "flavors" of hyperlink containers and targets.

1. Top-level documents.
2. OLE Document objects in a browser.
3. OLE Document objects in an Office Binder-like application that does not support OLE in-place/DocObj server functionality, and hence cannot be shown in-place in a browser.

The following are possible forms of navigation:

1. From one top-level document to another top-level document (for example, between Office documents in the absence of a browser).
2. From a top-level document to an OLE Document in a browser (for example, from a stand-alone application to an HTML document).
3. From one OLE document object in a browser to another OLE document object in the same browser (for example, one HTML document to another).
4. From one OLE document object in a browser to an OLE document object in a Binder-type application (for example, if the hyperlink target is embedded as an OLE Document Object in an Office Binder document; in this scenario, the end user model gets complicated and confusing if the embedded OLE Document Object is shown in the browser's window, instead of letting it appear in its own container's window).

5. From one location in an object/document to another location in the same object/document (applicable to all three flavors of hyperlink containers/hyperlink targets).

Methods in Vtable Order

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

IHlink Methods	Description
SetHlinkSite	Saves the interface pointer on the site object.
GetHlinkSite	Retrieves the interface pointer on the site object.
GetMonikerReference	Retrieves the moniker and location portions of the target.
GetStringReference	Retrieves the name and location portions of the target.
GetFriendlyName	Retrieves the friendly name of the target.
SetFriendlyName	Sets the friendly name of the target.
GetTargetFrameName	Retrieves the name of the target frame.
SetTargetFrameName	Sets the name of the target frame.
GetAdditionalParams	Retrieves additional hyperlink parameters.
SetAdditionalParams	Sets additional hyperlink parameters.
Navigate	Navigates to the given target.
GetMiscStatus	Queries if hyperlink is absolute or relative.

See Also

HlinkNavigateToStringReference, HlinkSimpleNavigateToMoniker, HlinkSimpleNavigateToString

IHlink::GetAdditionalParams

Retrieves additional parameters or properties for the hyperlink.

HRESULT GetAdditionalParams(
  LPWSTR * *pszAdditionalParams* // Extensible parameter string
);

Parameters

pszAdditionalParams
  [out] On return, points to the location to return the string containing a list of parameters or properties for the hyperlink in the following format:

```
<ID (1) = "value (1) "> <ID (2) = "value (2) "> ... <ID (n) = "value (n) ">
```

Return Values

S_OK
  The hyperlink's additional parameters string was retrieved.

Remarks

The parameters retrieved in this string are primarily interpreted by the hyperlink frame.

See Also

IHlink:SetAdditionalParams

IHlink::GetFriendlyName

Retrieves the friendly name of the target. The container normally uses this name to determine how to represent the hyperlink in its user interface.

```
HRESULT GetFriendlyName(
    DWORD grfHLFNAMEF,          // TBD
    LPWSTR * pszFriendlyName    // Receives the friendly name of the hyperlink target.
  );
```

Parameters

grfHLFNAMEF
[in] TBD.

pszFriendlyName
[out] On return, points to the friendly name of the hyperlink target. May not be NULL.

Return Values

S_OK
The friendly name string of the hyperlink was retrieved.

See Also

IHlink::SetFriendlyName, IHlinkTarget::GetFriendlyName

IHlink::GetHlinkSite

Retrieves the hyperlink site and associated site data from a hyperlink object.

```
HRESULT GetHlinkSite(
    IHlinkSite ** pphlSite,      // Receives the hyperlink site interface
    DWORD *pdwSiteData           // Receives additional site data
  );
```

Parameters

pphlSite

[out] On return, points to the IHlinkSite interface pointer on the hyperlink object. This may not be a NULL pointer on input.

pdwSiteData

[in, out] On return, points to the site data of the hyperlink target. This may not be a NULL pointer on input.

Return Values

S_OK

The hyperlink site and associated data have been successfully retrieved.

See Also

IHlink::SetHlinkSite

IHlink::GetMiscStatus

Queries whether or not the hyperlink is an absolute or a relative link.

```
HRESULT GetMiscStatus(
    DWORD * pdwStatus      // Receives absolute/relative status of hyperlink
  );
```

Parameters

pdwStatus
[out] On return, points to a value from the HLINKMISC enumeration. May not be NULL.

Return Values

S_OK
The hyperlink status has been successfully retrieved.

See Also

HLINKMISC

IHlink::GetMonikerReference

Retrieves the moniker and location portions of the hyperlink reference.

```
HRESULT GetMonikerReference(
    DWORD dwWhichRef,       // Absolute or relative reference
    IMoniker ** ppmk,       // Receives a pointer to the moniker of the hyperlink target
    LPWSTR * pszLocation    // Receives the location portion of the hyperlink target
  );
```

Parameters

dwWhichRef

[in] Value from the HLINKGETREF enumeration specifying whether to get the absolute or relative reference to the hyperlink target.

ppmk

[out] On return, points to the **IMoniker** pointer of the target's moniker object, if any. May be NULL, in which case the caller is not interested in the moniker to the hyperlink reference.

pszLocation

[in,out,unique] On return, points to the string location portion of the target, if any. If NULL, the caller is not interested in the location portion of the hyperlink reference.

Return Values

S_OK

Indicates the moniker and location portions of the target of the hyperlink target have been successfully supplied.

See Also

HLINKGETREF, IHlinkSite::GetMoniker, IHlinkTarget::GetMoniker

IHlink::GetStringReference

Retrieves strings that identify the hyperlink target and the location within the hyperlink target.

```
HRESULT GetStringReference(
  DWORD dwWhichRef,      // Absolute or relative reference
  LPWSTR * pszTarget,    // Receives the string that helps identify the hyperlink target
  LPWSTR * pszLocation   // Receives the location portion of the hyperlink target
);
```

Parameters

dwWhichRef

[in] Value from the HLINKGETREF enumeration specifying whether to get the absolute or relative reference to the hyperlink target.

pszTarget

[out] On return, points to a string that helps identify the hyperlink target of the hyperlink reference. If NULL, the caller is not interested in the target string of the hyperlink reference.

pszLocation

[out] On return, points to the location portion of the hyperlink reference. If NULL, the caller is not interested in the location portion of the hyperlink reference.

Return Values

S_OK

Indicates the target and location strings of the hyperlink target have been successfully supplied.

See Also

IHlink::GetMonikerReference, HLINKGETREF, HlinkNavigateToStringReference, HlinkSimpleNavigateToString

IHlink::GetTargetFrameName

Retrieves the name of the target frame for the hyperlink.

```
HRESULT GetTargetFrameName(
    LPWSTR * pszTargetFrameName      // Receives the target frame name for the hyperlink
  );
```

Parameters

pszTargetFrameName

[out] On return, points to the target frame name for the hyperlink. May not be NULL.

Return Values

S_OK

The target frame name string for the hyperlink was retrieved.

See Also

IHlink::SetTargetFrameName

IHlink::Navigate

Navigates to the hyperlink reference. **IHlink::Navigate** is the heart of the navigation process that implements the action of resolving a hyperlink target.

```
HRESULT Navigate(
    DWORD grfHLNF,                  // Navigation flags
    IBindCtx * pbc,                 // Bind context interface pointer
    IBindStatusCallback * pbsc,     // Bind-status-context interface pointer
    IHlinkBrowseContext * phlbc     // Browse context interface pointer
  );
```

Parameters

grfHLNF

[in] Flags describing how the navigation is to proceed. The value of the flag can be any valid HLNF enumeration value.

pbc

[in] **IBindCtx** interface pointer to the bind context to be used for any moniker binding during this operation. May not be NULL.

pbsc

[in] IBindStatusCallback interface pointer to the bind status context to use for any asynchronous moniker binding performed during the navigation. If NULL, the caller is not interested in progress notification, cancellation, pausing, or low-level binding information

phlbc

[in] Pointer to the IHlinkBrowseContext interface to use for this navigation. May not be NULL. As part of navigation, this browse context's navigation stack may be updated (depending on the navigation flags in *grfHLNF*) and its cache of hyperlink targets is consulted for a matching to the current hyperlink target.

Return Values

S_OK

Navigation to the hyperlink has been completed successfully.

HLINK_S_NAVIGATEDTOLEAFNODE

See Also

HlinkNavigate, HLNF, IHlinkFrame::Navigate, IHlinkTarget::Navigate

IHlink::SetAdditionalParams

Sets the additional parameters or properties for the hyperlink.

HRESULT SetAdditionalParams(
  LPCWSTR *szAdditionalParams* // Extensible parameter string
);

Parameters

szAdditionalParams
  [in] The string containing a list of parameters or properties for the hyperlink in the following format:

```
<ID (1) = "value (1) "> <ID (2) = "value (2) "> ... <ID (n) = "value (n) ">
```

Return Values

S_OK
  The hyperlink's additional parameters string was set.

Remarks

The parameters saved in this string are primarily interpreted by the hyperlink frame.

See Also

IHlink::GetAdditionalParams

IHlink::SetFriendlyName

Sets the *friendly name* for the hyperlink. The friendly name is used by hyperlink containers to represent the hyperlink within their user interface.

```
HRESULT SetFriendlyName(
    LPCWSTR szFriendlyName      // Friendly name of the hyperlink
  );
```

Parameters

szFriendlyName
[in]The friendly name string of the hyperlink target.

Return Values

S_OK
The friendly name string of the hyperlink was set.

See Also

IHlink::GetFriendlyName

IHlink::SetHlinkSite

Sets the hyperlink site and associated site data on a hyperlink object.

```
HRESULT SetHlinkSite(
    IHlinkSite * phlSite,       // Interface pointer to the new hyperlink site
    DWORD dwSiteData            // Additional site data
  );
```

Parameters

phlSite
[in] Pointer to a new IHlinkSite object for this hyperlink.

dwSiteData
[in] DWORD that specifies additional information about the site. For instance, each site may differ in color or style from other sites in the same container.

Return Values

S_OK
The hyperlink site object and its site data have been successfully set.

See Also

IHlink::GetHlinkSite

IHlink::SetTargetFrameName

Sets the target frame name for the hyperlink.

```
HRESULT SetTargetFrameName(
    LPCWSTR  szTargetFrameName      // Target frame name for the hyperlink
  );
```

Parameters

szTargetFrameName
[in] The string target frame name for the hyperlink.

Return Values

S_OK
The target frame name string for the hyperlink was set.

See Also

IHlink::GetTargetFrameName

IHlinkBrowseContext

The **IHlinkBrowseContext** interface enables a COM object to define and manage the browsing context for an OLE hyperlink application.

A hyperlink browse context keeps track of the order in which the object/documents (perhaps belonging to different processes) have been visited and chains them together into a navigation stack. The context also keeps track of enabling or disabling buttons like *Go Forward* or *Go Back* for each entry in the navigation stack. In addition, the context maintains window positioning information for hyperlink targets so that a hyperlink application can provide a seamless user-interface.

Browse context objects are not required for hyperlink navigation. HlinkSimpleNavigateToString and HlinkSimpleNavigateToMoniker provide examples of navigation between hyperlink containers and hyperlink targets that do not use a browsing context.

IHlinkBrowseContext provides methods that maintain the hyperlink navigation stack and keep track of window position information.

When to Implement

The system provides a default implementation.

Methods in Vtable Order

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

IHlinkBrowseContext Methods	Description
Register	Registers an object with the browse context.
GetObject	Retrieves an object previously registered in the browse context under the given name.
Revoke	Revokes an object previously registered with this browse context.
SetBrowseWindowInfo	Establishes the browse window info structure of this browse context.
GetBrowseWindowInfo	Retrieves the browse window info structure associated with this browse context.
EnumNavigationStack	Retrieves an enumerator which can be used to enumerate the current contents of the navigation stack.
QueryHlink	Tests the validity of a hyperlink ID value.
GetHlink	Retrieves a hyperlink from this browse context.
SetCurrentHlink	Sets the current hyperlink in this browse context's navigation stack.

OnNavigateHlink	Notifies a browse context that a hyperlink has been navigated.
Clone	Duplicates a browse context.
Close	Closes the hyperlink browse context.

See Also

HlinkCreateBrowseContext, IHlink, IHlinkFrame, IHlinkTarget

IHlinkBrowseContext::Clone

Duplicates a browse context.

```
HRESULT Clone(
    IUnknown* punkOuter,      // Controlling IUnknown interface for possible aggregation
    REFIID riid,              // Interface ID to return on the new browse context
    void** ppv                // Receives the requested interface's pointer
  );
```

Parameters

punkOuter

[in] Pointer to the controlling **IUnknown** interface for the new hyperlink object. If NULL, the new hyperlink object is not aggregated.

riid

[in] The interface to return on the new hyperlink. Typically IID_IHlink, or IID_IUnknown when *punkOuter* is non-NULL.

ppv

[out] Receives the pointer to the *riid* interface.

Return Values

S_OK

The browse context has been duplicated.

IHlinkBrowseContext::Close

Closes the hyperlink browse context. Releases all hyperlink targets that have been registered with the browse context.

```
HRESULT Close(
    DWORD dwReserved      // Reserved for future use
  );
```

Parameters

dwReserved

[in] Reserved for future use; must be set to zero by the caller. To ensure compatibility with future use, the callee *must not* check for zero.

Return Values

S_OK

The hyperlink browse context has been successfully closed.

See Also

IHlinkBrowseContext::Register

IHlinkBrowseContext::EnumNavigationStack

Retrieves an enumerator which can be used to enumerate the current contents of the navigation stack.

```
HRESULT EnumNavigationStack(
    IEnumHLITEM ** ppenumhlitem      // Location to receive the enumeration interface pointer
  );
```

Parameters

ppenumhlitem
[out] Location to return the IEnumHLITEM enumeration interface over the set of hyperlinks in this navigation stack.

Return Values

S_OK
The enumerator has been successfully retrieved.

See Also

HLID, HLITEM, IEnumHLITEM

IHlinkBrowseContext::GetBrowseWindowInfo

Retrieves the HLBWINFO structure currently associated with this browse context.

HRESULT GetBrowseWindowInfo(
  HLBWINFO * *phlbwi* // Location to receive the **HLBWINFO** structure
);

Parameters

phlbwi
  [out] Pointer to the location to return the current **HLBWINFO** structure.

Return Values

S_OK
  The **HLBWINFO** structure for this browse window has been retrieved.

See Also

HLBWINFO, IHlinkBrowseContext::SetBrowseWindowInfo

IHlinkBrowseContext::GetHlink

Retrieves a hyperlink from this browse context.

```
HRESULT GetHlink(
    ULONG uHLID,      // Hyperlink to retrieve
    IHlink ** pphl    // Buffer to return the IHlink interface of the hyperlink
  );
```

Parameters

uHLID

[in] Identifies the hyperlink to retrieve. May be a value taken from the HLID constants to indicate a logically identified hyperlink, such as HLID_PREVIOUS or HLID_NEXT.

pphl

[out] Pointer to the buffer to return the IHlink interface pointer of the hyperlink object.

Return Values

S_OK

The hyperlink has been successfully retrieved from this browse context.

E_FAIL

Indicates the specified hyperlink does not exist in this browse context.

See Also

HLID, IHlink, IHlinkBrowseContext::SetCurrentHlink

IHlinkBrowseContext::GetObject

Retrieves an object previously registered in the browse context.

```
HRESULT GetObject(
    IMoniker * pmk,          // Moniker interface pointer of the object being retrieved
    IUnknown ** ppunk        // Location to receive the unknown interface pointer of    requested object
  );
```

Parameters

pmk
[in,unique] Identifies the object being retrieved.

ppunk
[out] Location to return the **IUnknown** interface of the object being retrieved.

Return Values

S_OK
Indicates the previously registered object has been retrieved.

S_FALSE
Indicates no object indicated by *pmk* was registered in the browse context.

See Also

IHlinkBrowseContext::Register

IHlinkBrowseContext::OnNavigateHlink

Notifies a browse context that a hyperlink has been navigated.

```
HRESULT OnNavigateHlink(
    DWORD grfHLNF,              // Navigation flags
    IMoniker * pmkTarget,       // Moniker interface pointer of the hyperlink target
    LPCWSTR pszLocation,        // Location within the hyperlink target of new hyperlink
    LPCWSTR szFriendlyName      // Friendly name of the hyperlink
  );
```

Parameters

grfHLNF

[in] Flags describing how the navigation is to proceed. The value of the flag can be any valid HLNF enumeration value.

pmkTarget

[in] Pointer to an **IMoniker** interface on the hyperlink target.

pszLocation

[in] A string identifying the location within the hyperlink target that was navigated to. May not be NULL.

szFriendlyName

[in] The friendly name of the location within the hyperlink target that has been navigated to. May not be NULL.

Return Values

S_OK

The browse context has been successfully notified that a hyperlink has been navigated.

See Also

HlinkOnNavigate, HLNF, IHlinkFrame::OnNavigate, IHlinkSite::OnNavigationComplete

IHlinkBrowseContext::QueryHlink

Tests the validity of a uHLID value.

```
HRESULT QueryHlink(
    DWORD grfHLQF,      // Value from the HLQF enumeration
    ULONG uHLID         // Hyperlink identifier to query
  );
```

Parameters

grfHLQF

[in] A single value taken from the HLQF enumeration.

uHLID

[in] Identifies the hyperlink to query about. May be a value taken from the **HLID** constants to indicate a logically identified hyperlink, such as HLID_PREVIOUS or HLID_NEXT.

Return Values

S_OK

If *grfHLQF* is HLQF_ISVALID, uHLID identifies a valid hyperlink within the browse context. If *grfHLQF* is HLQF_ISCURRENT, uHLID identifies the current hyperlink of the browse context.

S_FALSE

If *grfHLQF* is HLQF_ISVALID, uHLID does not identify a valid hyperlink within the browse context. If *grfHLQF* is HLQF_ISCURRENT, uHLID does not identify the current hyperlink of the browse context.

See Also

HLID, HLQF

IHlinkBrowseContext::Register

Registers an object with the browse context.

```
HRESULT Register(
    DWORD dwReserved,       // Reserved for future use
    IUnknown punk,          // Object being registered
    IMoniker * pmk,         // Moniker interface pointer of the object being registered
    DWORD * pdwRegister     // Receives the registration value
  );
```

Parameters

dwReserved

[in] Reserved for future use; must be set to zero by the caller. To ensure compatibility with future use, the callee *must not* check for zero.

punk

[in, unique] The object being registered.

pmk

[in, unique] Pointer to the **IMoniker** interface pointer that identifies the object being registered.

pdwRegister

[out] Pointer to a location to return a value identifying the registration which can be used to subsequently revoke the registration.

Return Values

S_OK

The object has been registered.

MK_S_MONIKERALREADYREGISTERED

The object was successfully registered, but another object (possibly the same object) has already been registered with the same moniker in this browse context.

E_OUTOFMEMORY

There was insufficient memory to register the object with the browse context.

See Also

HlinkCreateBrowseContext, IHlinkBrowseContext::Close, IHlinkBrowseContext::Revoke

IHlinkBrowseContext::Revoke

Revokes an object previously registered with this browse context.

HRESULT Revoke(
  DWORD *dwRegister* // Object to revoke
);

Parameters

dwRegister
  [in] Identifies the object to revoke. Returned by a previous call to **IHlinkBrowseContext::Register**.

Return Values

S_OK
  The previously-registered object has been revoked from this browse context.

See Also

IHlinkBrowseContext::Register

IHlinkBrowseContext::SetBrowseWindowInfo

Establishes the HLBWINFO structure of this browse context.

HRESULT SetBrowseWindowInfo(
  HLBWINFO * *phlbwi* // Pointer to the new **HLBWINFO** structure for this browse context
);

Parameters

phlbwi
  [in,unique] Pointer to the new **HLBWINFO** structure for this browse context.

Return Values

S_OK
  The **HLBWINFO** structure for this browse window has been established.

See Also

HLBWINFO, IHlinkBrowseContext::GetBrowseWindowInfo

IHlinkBrowseContext::SetCurrentHlink

Sets the current hyperlink in this browse context's navigation stack.

HRESULT SetCurrentHlink(
  ULONG *uHLID* // Hyperlink ID to set in the navigation stack
);

Parameters

uHLID
  [in] Identifies the hyperlink to set in the current browse context's navigation stack. May be a value taken from the **HLID** constants to indicate a logically identified hyperlink, such as HLID_PREVIOUS or HLID_NEXT.

Return Values

S_OK
  This hyperlink has been successfully set in the current browse context's navigation stack.

E_FAIL
  Indicates the specified hyperlink does not exist in this browse context.

See Also

HLID, IHlinkBrowseContext::GetHlink

IHlinkFrame

The **IHlinkFrame** interface provides methods that allow a hyperlink frame to interpose itself in the navigation process to allow an application to provide a seamless user interface when navigating from one document/object/application to another when resolving a hyperlink.

When to Implement

When to Use

You do not normally call the **IHlinkFrame** methods directly. They are referenced by the IHlink::Navigate routines when a hyperlink frame is detected.

Methods in Vtable Order

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

IHlinkFrame Methods	Description
GetBrowseContext	Retrieves the browse context of the hyperlink frame
Navigate	Navigates to the target hyperlink
OnNavigate	Notifies the hyperlink frame of a hyperlink navigation
SetBrowseContext	Sets the browse context of the hyperlink frame

See Also

HlinkNavigate, HlinkNavigateToStringReference, HlinkSimpleNavigateToMoniker, HlinkSimpleNavigateToString, IHlink

IHlinkFrame::GetBrowseContext

Retrieves the browse context of the hyperlink frame.

```
HRESULT GetBrowseContext(
    IHlinkBrowseContext ** pphlbc      // Buffer to receive the interface of the current browse context
  );
```

Parameters

pphlbc
[out] Location to return the browse context interface's pointer.

Return Values

S_OK
The browse context has been successfully retrieved.

E_NOTIMPL
The hyperlink target does not understand browse contexts.

See Also

HlinkCreateBrowseContext, IHlinkBrowseContext, IHlinkFrame::SetBrowseContext, IHlinkTarget

IHlinkFrame::Navigate

Navigates to the hyperlink target.

```
HRESULT Navigate(
    DWORD grfHLNF,              // How to navigate
    IBindCtx * pbc,             // Bind context interface pointer
    IBindStatusCallback * pbsc, // Bind-status callback   interface pointer
    IHlink * phlNavigate        // Hyperlink target interface pointer
  );
```

Parameters

grfHLNF

[in] Flags describing how the navigation is to proceed. The value of the flag can be any valid HLNF enumeration value.

pbc

[in] Pointer to the **IBindCtx** object interface to use for any moniker binding during this operation. May not be NULL.

pbsc

[in] Pointer to the IBindStatusCallback object interface to use for any asynchronous moniker binding performed during the navigation. If NULL, the caller is not interested in progress notification, cancellation, pausing, or low-level binding information.

phlNavigate

[in] Pointer to the IHlink object interface to the hyperlink target.

Return Values

S_OK

The navigation to the target has been successful.

IHlink::Navigate errors

Since **IHlinkFrame::Navigate** calls **IHlink::Navigate**, the errors associated with this method can be returned.

See Also

HLNF, IHlink::Navigate

IHlinkFrame::OnNavigate

Notifies the hyperlink frame that a hyperlink has been navigated to.

HRESULT OnNavigate(
  DWORD *grfHLNF* // Navigation flags
);

Parameters

grfHLNF
  [in] Flags describing how the navigation is to proceed. The value of the flag can be any valid **HLNF** enumeration value.

Return Values

S_OK
  The hyperlink frame has been successfully notified.

See Also

HlinkOnNavigate, HLNF, IHlinkBrowseContext::OnNavigateHlink, IHlinkSite::OnNavigationComplete

IHlinkFrame::SetBrowseContext

Sets the browse context of the hyperlink frame.

```
HRESULT GetBrowseContext(
    IHlinkBrowseContext * phlbc      // Interface pointer of the current browse context
  );
```

Parameters

phlbc
[in] The browse context interface's pointer.

Return Values

S_OK
The browse context has been successfully set.

E_NOTIMPL
The hyperlink target does not understand browse contexts.

See Also

HlinkCreateBrowseContext, IHlinkBrowseContext, IHlinkFrame::SetBrowseContext, IHlinkTarget::SetBrowseContext

IHlinkSite

The **IHlinkSite** interface provides methods for a *hyperlink* to retrieve the moniker or the interface on its *hyperlink container*. The navigation process which resolves a hyperlink uses this information to efficiently handle internal (within the same container) and external (to a different container) hyperlink references.

When to Implement

Implement **IHlinkSite** when you want to hyperlink to other documents. For each hyperlink object provided by the system, a third party, or that you create, you provide an **IHlinkSite** object. One site object can support multiple hyperlinks.

Methods in Vtable Order

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

IHlinkSite Methods	Description
GetMoniker	Retrieves the moniker to the hyperlink target.
GetInterface	Retrieves the interface to the hyperlink target.
OnNavigationComplete	Notifies the hyperlink site that hyperlink navigation is complete.

See Also

IHlink

IHlinkSite::GetInterface

Retrieves the hyperlink target interface on a hyperlink's container (usually the document that contains the hyperlink site).

```
HRESULT GetInterface(
    DWORD dwSiteData,       // Site data to identify the hyperlink its site
    DWORD dwReserved,       // Reserved for future use
    REFIID riid,            // Interface ID to return on the hyperlink container
    Void* ppv               // Buffer to receive the riid interface
  );
```

Parameters

dwSiteData

[in] Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.

dwReserved

[in] Reserved for future use; must be set to zero by the caller. To ensure compatibility with future use, the callee *must not* check for zero.

riid

[in] Identifies the interface to return.

ppv

[out] Location to return the *riid* interface.

Return Values

S_OK

The hyperlink container interface was successfully retrieved.

E_NOINTERFACE

The desired interface is not available.

See Also

IHlinkSite::GetMoniker

IHlinkSite::GetMoniker

Returns the relative moniker of the hyperlink's container.

```
HRESULT GetMoniker(
    DWORD dwSiteData,      // Site data for the new hyperlink object
    DWORD dwAssign,        // Whether or not to create a moniker if one not found
    DWORD dwWhich,         // Which moniker to retrieve
    IMoniker ** ppmk       // Buffer to receive the requested moniker interface pointer
  );
```

Parameters

dwSiteData

[in] Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.

dwAssign

[in] A value from the OLEGETMONIKER enumeration. This is typically either OLEGETMONIKER_ONLYIFTHERE, indicating the function should not force a moniker to be created if one does not already exist, or OLEGETMONIKER_FORCEASSIGN, indicating that the function should create a moniker if one does not exist.

dwWhich

[in] A value from the OLEWHICHMK enumeration. This is typically OLEWHICHMK_CONTAINER, indicating that the site should return the moniker of the hyperlink container.

ppmk

[out] Location to return the **IMoniker** interface.

Return Values

S_OK

Indicates the moniker of the hyperlink's site container was successfully supplied.

E_FAIL

Indicates a moniker does not exist for this hyperlink target and OLEGETMONIKER_ONLYIFTHERE was specified for *dwAssign.*

See Also

IHlink::GetMonikerReference, IHlinkSite::GetInterface, IHlinkTarget::GetMoniker

IHlinkSite::OnNavigationComplete

Notifies the hyperlink site that a hyperlink navigation has been successful. This notification in particularly useful if the hyperlink has been navigated asynchronously because it is the only notification the hyperlink receives to realize that hyperlinking has completed.

```
HRESULT OnNavigationComplete(
    DWORD dwSiteData,       // Site data for the hyperlink object
    HRESULT hrStatus,       // Result of the hyperlink navigation
    LPCWSTR pszStatus       // Description of the failure, if any
);
```

Parameters

dwSiteData

[in] Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.

hrStatus

[in] Result of the hyperlink navigation. Either S_OK for success or E_ABORT or E_FAIL for failure.

pszStatus

[in] Pointer to the string describing the failure that occurred.

Return Values

S_OK

The hyperlink site has been successfully notified.

See Also

HlinkOnNavigate, IHlinkBrowseContext::OnNavigateHlink, IHlinkFrame::OnNavigate

IHlinkTarget

A hyperlink target implements the **IHlinkTarget** interface and gives out hyperlink references to its contents.

When to Implement

Implement this interface when you have an application that can be the target for a hyperlink. In other words, hyperlinks from other applications use this interface to determine how to find, display, and navigate to your target.

An existing OLE document application which supports OLE linking need only implement the **IHlinkTarget** interface on the same object that implements **IPersistFile** and **IOleItemContainer**. The application may also implement IPersistMoniker to support incremental rendering or asynchronous download as a persistence mechanism, rather than **IPersistFile**.

When to Use

You do not normally call these methods directly. When an end user selects a hyperlink through an application's user interface, the implementation of IHlink::Navigate calls these methods to determine how to display, locate, and navigate to the target.

Methods in Vtable Order

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

IHlinkTarget Methods	Description
SetBrowseContext	Establishes the current browse context for this hyperlink target.
GetBrowseContext	Retrieves the current browse context within which this hyperlink target is currently running.
Navigate	If the given location is not visible, navigates to and shows that position within the target.
GetMoniker	Returns a moniker to the hyperlink target object.
GetFriendlyName	Returns a friendly name for the given hyperlink location within this target.

See Also

HlinkNavigate, HlinkSimpleNavigateToMoniker, HlinkSimpleNavigateToString, IHlink, IHlinkBrowseContext

IHlinkTarget::GetBrowseContext

Retrieves the browse context in which this hyperlink target is currently running.

```
HRESULT GetBrowseContext(
    IHlinkBrowseContext ** pphlbc      // Buffer to receive the current browse context's interface pointer
  );
```

Parameters

pphlbc
[out] Pointer to the location to return the browse context object's interface pointer.

Return Values

S_OK
The browse context has been successfully retrieved.

E_NOTIMPL
The hyperlink target does not understand browse contexts.

See Also

HlinkCreateBrowseContext, IHlinkBrowseContext, IHlinkFrame::GetBrowseContext, IHlinkTarget::SetBrowseContext

IHlinkTarget::GetFriendlyName

Retrieves a friendly name for the given hyperlink destination within this target.

```
HRESULT GetFriendlyName(
    LPCWSTR szLocation,          // Hyperlink location within this target
    LPWSTR * pszFriendlyName     // Buffer to receive the friendly name
  );
```

Parameters

szLocation

[in, unique] String that indicates the position of the hyperlink destination within this hyperlink target.

pszFriendlyName

[out] Pointer to the buffer to return the friendly name. This string must be allocated using **CoTaskMemAlloc**. It is the caller's responsibility to free this string using **CoTaskMemFree**.

Return Values

S_OK

The friendly name string of the hyperlink destination with this target has been retrieved.

E_OUTOFMEMORY

Indicates there is insufficient memory to retrieve the friendly name.

See Also

IHlink::GetFriendlyName

IHlinkTarget::GetMoniker

Returns a moniker to the hyperlink target object for the given hyperlink destination *szLocation*.

```
HRESULT GetMoniker(
    LPCWSTR szLocation,         // Hyperlink destination within this target
    DWORD dwAssign,             // Whether or not to create a moniker if one not found
    IMoniker   ** ppmkLocation  // Buffer to receive this moniker
  );
```

Parameters

szLocation

[in, unique] Identifies the hyperlink destination within this target.

dwAssign

[in] A value from the OLEGETMONIKER enumeration. This is typically either OLEGETMONIKER_ONLYIFTHERE, indicating the function should not force a moniker to be created if one does not already exist, or OLEGETMONIKER_FORCEASSIGN, indicating that the function should create a moniker if one does not exist.

ppmkLocation

[out] Pointer to location to return the **IMoniker** interface pointer to the hyperlink target object for this *szLocation*.

Return Values

S_OK

The moniker to the hyperlink target object retrieved.

E_FAIL

A moniker does not exist for this hyperlink target and OLEGETMONIKER_ONLYIFTHERE was specified for *dwAssign.*

CreateFileMoniker, MkParseDisplayName, and other moniker creation API errors

See Also

HlinkSimpleNavigateToMoniker, IHlink::GetMonikerReference, IHlinkSite::GetMoniker

IHlinkTarget::Navigate

If the given location is not visible, navigates to and shows the location position within the object/document.

```
HRESULT Navigate(
    DWORD grfHLNF,          // Value from the HLNF enumeration
    LPCWSTR * szLocation    // Location within the hyperlink target to navigate to
);
```

Parameters

grfHLNF

[in] Flags describing how the navigation is to proceed. The value of the flag can be any valid **HLNF** enumeration value.

szLocation

[in] The string location portion of the hyperlink reference.

Return Values

S_OK

The navigation to the hyperlink target has been successful.

See Also

HlinkNavigate, HlinkNavigateToStringReference, HLNF, IHlinkFrame::Navigate, IHlink::Navigate

IHlinkTarget::SetBrowseContext

Establishes the current browse context for this hyperlink target.

```
HRESULT SetBrowseContext(
    IHlinkBrowseContext * phlbc      // Browse context interface pointer to set for the hyperlink target
  );
```

Parameters

phlbc
[in, unique] Pointer to the browse context object's interface.

Return Values

S_OK
The browse context has been successfully set.

E_NOTIMPL
The hyperlink target does not understand browse contexts.

See Also

HlinkCreateBrowseContext, IHlinkBrowseContext, IHlinkFrame::SetBrowseContext, IHlinkTarget::GetBrowseContext

IHttpNegotiate

During a bind operation, the moniker can obtain some additional bind information by calling the bind client's **IBindStatusCallback::GetBindInfo**. In some cases, additional callback interfaces can be provided by the client requesting the bind operation to supply even more services to the moniker performing the bind. These additional interfaces are typically requested by the moniker calling **IBindStatusCallback::QueryInterface** for the interface desired. However, a moniker client can also provide additional interfaces through **IServiceProvider**.

In the case of URL monikers, the **IHttpNegotiate** interface is an additional callback interface that the client can provide.

When to Implement

Clients of URL monikers that want to participate in HTTP negotiations that take place when binding to HTTP URLs can implement this interface. The clients can then add headers to HTTP requests and can examine HTTP response headers.

Note that not all header types are currently supported. Specifically, clients should not add Content Length or Accept headers to HTTP requests. Instead of adding Accept headers, the client can call **RegisterFormatEnumerator** to specify accepted types for a bind operation.

Note that this callback interface can be provided by multiple clients of an HTTP bind operation. Each client can participate in the HTTP negotiation process by looking at existing request headers and adding new ones.

When to Use

The URL moniker calls this interface when bind operation requires HTTP negotiation. To get a pointer to this interface, the moniker calls the client's **IBindStatusCallback::QueryInterface**.

Methods in Vtable Order

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

IHttpNegotiate Methods	Description
BeginningTransaction	Notifies a client of a URL being bound to at the beginning of the HTTP transaction.
OnResponse	Notifies a client upon receiving a response to an HTTP request.

See Also

IWindow, IWinINetInfo

IHttpNegotiate::BeginningTransaction

Notifies the client of the URL being bound at the beginning of an HTTP transaction.

```
HRESULT BeginningTransaction(
    LPCWSTR   szURL,                //URL for the HTTP transaction
    DWORD     dwReserved,           //Reserved for future use
    LPCWSTR   szHeaders,            //Current request headers
    LPWSTR *pszAdditionalHeaders    //Optional additional headers to append to the HTTP request header
  );
```

Parameter

szURL
[in] URL for the HTTP transaction.

dwReserved
[in] Reserved for future use.

szHeaders
[in] Current request headers.

pszAdditionalHeaders
[out] Optional additional headers to append to the HTTP request. If these conflict with existing values in *szHeaders*, then the new request headers take precedence. If *pszAdditionalHeaders* is set to NULL, no headers are added to the HTTP request.

Return Values

S_OK
The operation was successful. Any additional headers specfied are appended.

E_INVALIDARG
The parameter is invalid.

E_ABORT
Terminate the HTTP transaction.

Remarks

The URL moniker calls **IHttpNegotiate::BeginningTransaction** before sending an HTTP request. It notifies the client of the URL being bound to at the beginning of the HTTP transaction. It also allows the client to add additional headers, such as Accept-Language, to the request.

See Also

IHttpNegotiate::OnResponse

IHttpNegotiate::OnResponse

Allows the client of a bind operation to examine the response headers, optionally, terminate the bind operation, and add HTTP headers to a request before resending the request.

```
HRESULT OnResponse(
    DWORD   dwResponseCode,                 //HTTP response code received
    LPCWSTR   szResponseHeaders,            //Response headers from the HTTP server
    LPCWSTR   szRequestHeaders,             //HTTP headers provided by the bind clients
    LPWSTR *pszAdditionalResquestHeaders    //Optional additional HTTP headers
  );
```

Parameter

dwResponseCode
[in] HTTP response code returned in response to a previous HTTP request. See the HTTP specification for information about these codes.

szResponseHeaders
[in] Response headers from the HTTP server. See the HTTP specification for more information.

szRequestHeaders
[in] If *dwResponseCode* indicates an error, contains HTTP request headers that will be used when the request is sent again.

pszAdditionalRequestHeaders
[out] If *dwResponseCode* indicates an error, contains optional, additional headers to add before the request is sent again. If the specified header value conflicts with existing values in *szRequestHeaders*, then the new headers take precedence. If *pszAdditionalRequestHeaders* is set to NULL, no headers are added to the HTTP request.

Return Values

S_OK
The operation was successful. If the *dwResponseCode* indicates an error, any *szAdditionalRequestHeaders* are appended and the request is sent again.

E_INVALIDARG
The parameter is invalid.

E_ABORT
Terminate the HTTP transaction.

Remarks

The URL moniker calls this method when it has received a response to an HTTP request. If the *dwResponseCode* indicates a success, the client can examine the response headers and can optionally abort the bind operation. If the *dwResponseCode* indicates a failure, the client can add HTTP headers to the request before it is sent again.

Note that if multiple clients have registered on the bind context for a given bind operation, more than one of these clients can provide an **IHttpNegotiate** callback interface. Every client providing this callback receives notifications. Each one is given a chance to add HTTP headers or to abort the HTTP transaction. In such cases, the last client to receive callback notification is the client driving the download operation, and it will dictate the final decision.

See Also

IHttpNegotiate::BeginningTransaction

IMoniker - URL Moniker Implementation

The URL moniker implementation of **IMoniker** is found on a URL moniker object, which also supports **IUnknown** and **IAsyncMoniker**. The **IMoniker** interface inherits its definition from **IPersistStream** as well as **IUnknown** and **IPersistStream** inherits from **IPersist**. Thus, the **IMoniker** implementation includes support for **IPersistStream** and **IPersist**.

The **IAsyncMoniker** interface is simply **IUnknown** (there are no additional methods). It is is used to allow clients to determine if a moniker supports asynchronous binding.

To get a pointer to the **IMoniker** interface on this object, call the CreateURLMoniker function.

When to Use

If you're a moniker client (that is, you're using a moniker to get an interface pointer to an object), you typically don't need to know the class of the moniker you're using; you simply call methods using an **IMoniker** interface pointer.

If you're a moniker provider (that is, you're handing out monikers that identify your objects to make them accessible to moniker clients), you must use item monikers if the objects you're identifying are contained within another object and can be individually identified using a string. You'll also need to use another type of moniker (for example, file monikers) in order to identify the container object.

To use item monikers, you must use the **CreateItemMoniker** function to create the monikers. To allow your objects to be loaded when an item moniker is bound, the container of your objects must implement the **IOleItemContainer** interface.

The most common example of moniker providers are OLE applications that support linking. If your OLE application supports linking to objects smaller than a file-based documents, you need to use item monikers. For a server application that allows linking to a selection within a document, you use the item monikers to identify those objects. For a container application that allows linking to embedded objects, you use the item monikers to identify the embedded objects.

Remarks

IMoniker::BindToObject

Since the URL Moniker supports asynchronous binding, the actual return value of its **BindToObject** may vary depending on the object parameters established in the bind-context, however the semantics of the bind operation are identical regardless of synchronous or asynchronous usage, and are as follows:

1. The URL Moniker pulls further information for the bind operation from the bind context. For example, the moniker can obtain pointers to the **IBindStatusCallback** and **IEnumFORMATETC** interfaces that are registered in the bind context. Note that further information can include additional bind options specified on the bind context through **IBindCtx::SetBindOptions**, such as the *dwTickCountDeadline* parameter or the *grfFlags* value of BIND_MAYBOTHERUSER.
2. Next, the moniker checks the Running Object Table of the bind-context to determine if the referenced object is already running. The moniker can obtain this information with the following calls:

   ```
   IBindCtx::GetRunningObjectTable(&prot)
   prot->IsRunning(this)
   ```

   If the object is already running, the moniker retrieves the running object with the following call:

   ```
   prot->GetObject(this, &punk)
   ```

   Then, the moniker calls **QueryInterface** for the requested interface.

3. Otherwise, the moniker queries the client by calling **IBindStatusCallback::GetBindInfo** to obtain additional bind information. The moniker then initiates the bind operation and passes the resulting **IBinding** interface back to the client by calling **IBindStatusCallback::OnStartBinding**.
4. If in step 1 it was determined that this was an asynchronous bind, **BindToObject** returns MK_S_ASYNCHRONOUS at this point with NULL in *ppv*. The caller will receive the actual object pointer during IBindStatusCallback::OnObjectAvailable at some later point. The following steps then occur asynchronously to the caller, typically on another thread of execution.
5. The class of the resource designated by the URL Moniker is determined in one of the following ways:

- The URL Moniker examines the media type of the data. If the media type is **application/x-oleobject**, the first 16-bytes of the actual data (*Content-Body*) contain the CLSID of the resource and subsequent data is to be interpreted by the class itself. For all other media types, URL Moniker looks in the system registry for the HKEY_CLASSES_ROOT\MIME\Database\Content-Type\<media-type>\CLSID key. Note that **application/x-oleobject** will be used until **application/oleobject** is approved.
- The URL Moniker matches portions of arriving data to patterns registered in the system registry under HKEY_CLASSES_ROOT\FileTypes.
- Finally, if all else fails, the URL Moniker correlates the trailing extension of the resource, if any, to a CLSID using the HKEY_CLASSES_ROOT\.??? keys in the system registry, as is done by **GetClassFile** and the shell.

6. Having determined the class, the URL moniker creates an instance using **CoCreateInstance** of CLSCTX_SERVER asking for the **IUnknown** interface.
7. The URL Moniker next calls the **QueryInterface** method of the newly created object for the **IPersistMoniker** interface. If the **QueryInterface** is successful, the URL Moniker calls IPersistMoniker::Load passing itself (**this**) as the moniker parameter. The object typically calls **IMoniker::BindToStorage** asking for the storage interface that they're interested in.
8. Otherwise, the URL Moniker calls the **QueryInterface** method for **IPersistStream** and, if successful, calls **IPersistStream::Load**, passing the object an **IStream** pointer for a stream object that is being filled asynchronously by the transport.

   If the class being called is not marked with the category CATID_AsyncAware, calls to **IStream::Read** or **IStream::Write** which reference data not yet available block until the data becomes available. These calls block in the traditional OLE sense. A message loop is entered which allows certain messages to be processed, and the **IMessageFilter** of the thread is called appropriately.

   If the class is marked with the category CATID_AsyncAware, calls to **IStream::Read** or **IStream::Write** which reference data not yet available return E_PENDING.
9. Otherwise, the URL Moniker calls **QueryInterface** for **IPersistFile**, and, if successful, completes the download into a temporary file. On completion, the URL Moniker calls **IPersistFile::Load**. The created file is cached along with other Internet downloaded data. The client must be sure not to delete this file.
10. When the object returns from one of the various **IPersist*Xxx*::Load** calls described in the previous steps, the URL Moniker calls **IBindStatusCallback::OnObjectAvailable** to return the interface pointer that the client originally requested when the client called **IMoniker::BindToObject**.

IMoniker::BindToStorage

The system implementation of URL Monikers supports the **BindToStorage** for stream objects on all URLs and for storage objects in the case where the designated resource is in fact a compound file. Future support for **ILockBytes** may also be added.

Since the URL Moniker supports asynchronous binding, the actual return value of its **BindToStorage** may vary depending on the object parameters established in the bind-context.

However, the semantics of the bind operation are identical regardless of synchronous or asynchronous usage as described below:

1. The URL Moniker pulls further information for the bind operation from the bind context. For example, the moniker can obtain pointers to the **IBindStatusCallback** and **IEnumFORMATETC** interfaces that are registered in the bind context. Note that further information can include additional bind options specified on the bind context through **IBindCtx::SetBindOptions**, such as the *dwTickCountDeadline* parameter or the *grfFlags* value of BIND_MAYBOTHERUSER. The moniker then queries the client by calling **IBindStatusCallback::GetBindInfo** and initiates the bind operation with the transport, and passes the resulting **IBinding** to the client by calling **IBindStatusCallback::OnStartBinding**.
2. If the caller requested an asynchronous **IStream** or **IStorage** by specifying the **BINDF_ASYNCSTORAGE** flag in the **BINDINFO** structure retrieved from IBindStatusCallback::GetBindInfo, the URL moniker returns the object as soon as possible. Calls to these **IStorage** or **IStream** objects which reference data not yet available return E_PENDING.
3. If the caller does not specify asynchronous **IStream** or **IStorage** as described above, the URL Moniker will still return an object through IBindStatusCallback::OnDataAvailable as soon as possible. However calls to these objects which reference data not yet available will block until the data becomes available. For some applications this will require the least modification of their existing I/O code, yet may still result in improved performance depending on their access-patterns.

IMoniker::Reduce

Returns MK_S_REDUCED_TO_SELF and itself (**this**) in **ppmkReduced*.

IMoniker::ComposeWith

URL Monikers support composition of two URLs: a base URL composed with a relative URL. This composition is done according to the RFC on relative URLs. URL Monikers do not currently support the composition of a base URL moniker with a relative file moniker, although this may be supported in the future.

However, URL Monikers do support generic composition. If *fOnlyIfNotGeneric*==*TRUE*, the **ComposeWith** method returns MK_E_NEEDGENERIC. Otherwise, this method simply returns **CreateGenericComposite**(**this**, **pmkRight**, **ppmkComposite**). See the Win32 documentation for more information about **IMoniker::ComposeWith**.

IMoniker::Enum

Returns S_OK and sets **ppenumMoniker* to NULL, indicating that the moniker does not contain sub-monikers.

IMoniker::IsEqual

Returns S_FALSE if the other moniker (*pmkOtherMoniker*) is not an URL moniker, which it checks using **IPersist::GetClassID** to see if the CLSID is **CLSID_URLMoniker**. If the other moniker is an URL moniker, it compares the display names of the monikers for equality, returning either S_OK if they are identical or S_FALSE if not.

IMoniker::Hash

Creates a hash value based on the URL string of the moniker. This hash value is identical when URL strings are identical, although it may also be identical for different URL strings. This method is used to speed up comparisons by reducing the amount of time that it is necessary to call **IMoniker::IsEqual**.

IMoniker::IsRunning

Returns S_OK if this moniker is currently running. Otherwise, it returns S_FALSE. The URL Moniker determines if it is running by first checking if it is equal to the newly running moniker by making the following call:

```
pmkNewlyRunning->IsEqual
```

Typically, this call an inexpensive operation. If this does not succeed, the moniker next checks to see if it is registered with the Running Object Table of the passed-in bind-context.

IMoniker::GetTimeOfLastChange
Returns the time of last change of an object that is registered in the running object table.

IMoniker::Inverse
Returns MK_E_NOINVERSE.

IMoniker::CommonPrefixWith
Currently returns E_NOTIMPL. In the future, it may properly compute the proper common prefix of two URL monikers. See the Win32 documentation about **IMoniker::CommonPrefixWith** for details.

IMoniker::RelativePathTo
Currently returns E_NOTIMPL. In the future, it may properly compute the relative path between two URL monikers. See the Win32 documentation about **IMoniker::RelativePathTo** for details.

IMoniker::GetDisplayName
THe URL Moniker attempts to return its full URL string. If the moniker was created with a partial URL string (see CreateURLMoniker), it will first attempt to find an URL moniker in the bind-context under SZ_URLCONTEXT, and will next look to the moniker to its left for contextual information. If it can not return its full URL string, it will return its partial URL string.

IMoniker::ParseDisplayName
Parses a full or partial URL string into a result moniker (*ppmkOut*). If *szDisplayName* represents a full URL string (i.e. “http://foo.com/default.html”), the result is a new full URL moniker. If *szDisplayName* represents a partial URL string (i.e. “..\default.html”), the result is a full URL that takes its context from either the bind-context’s SZ_URLCONTEXT object-parameter or from this URL moniker. For example, if the context moniker was “http://foo.com/pub/list.html” and *szDisplayName* was “..\default.html”, the resulting URL moniker would represent “http://foo.com/default.html”.

IMoniker::IsSystemMoniker
Returns S_TRUE and MKSYS_URLMONIKER in **pdwMksys*.

See Also
CreateURLMoniker, IPersistStream -URL Moniker Implementation

IObjectWithSite

Often an object needs to communicate directly with a container site, that is managing the object itself. Outside of **IOleObject::SetClientSite**, there is no generic means through which an object becomes aware of its site. **IObjectWithSite** provides simple objects with a lightweight means (lighter than **IOleObject**) with a siting mechanism. This interface should only be used when **IOleObject** is not already in use.

Through **IObjectWithSite**, a container can pass the **IUnknown** pointer of its site to the object through **IObjectWithSite::SetSite**. Callers can also retrieve the latest site passed to **IObjectWithSite::SetSite** through **IObjectWithSite::GetSite**. This latter method is included as a hooking mechanism, allowing a third party to intercept calls from the object to the site.

Methods in Vtable Order

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

IObjectWithSite Methods	Description
SetSite	Provides the site's **IUnknown** pointer to the object being managed.
GetSite	Retrieves the last site set with **IObjectWithSite::SetSite**.

IObjectWithSite::GetSite

Retrieves the last site set with IObjectWithSite::SetSite. If there's no known site, the object return a failure code.

```
HRESULT GetSite(
   REFIID   riid,        // IID of interface pointer being requested
   void   ** ppvSite     // Receives an indirect pointer to caller's void
  );
```

Parameters

riid

[in] The IID of the interface pointer that should be returned in *ppvSite*.

ppvSite

[out] The address of the caller's **void** * variable in which the object stores the interface pointer of the site last seen in IObjectWithSite::SetSite. The specific interface returned depends on the *riid* argument—in essence, the two arguments act identically to those in **IUnknown::QueryInterface**. If the appropriate interface pointer is available, the object must call **IUnknown::AddRef** on that pointer before returning successfully. If no site is available, or the requested interface is not supported, the object sets this argument to NULL and returns a failure code.

Return Values

S_OK

The site was returned successfully and the caller is responsible for calling **IUnknown::Release** when the site is no longer needed.

E_FAIL

There is no site in which case **ppvSite* contains NULL on return.

E_NOINTERFACE

There is a site but it does not support the interface requested by *riid*.

Remarks

E_NOTIMPL is not allowed—any object implementing this interface must be able to return the last site seen in IObjectWithSite::SetSite.

IObjectWithSite::SetSite

Provides the site's **IUnknown** pointer to the object. The object should hold onto this pointer, calling **IUnknown::AddRef** in doing so. If the object already has a site, it should call that existing site's **IUnknown::Release**, save the new site pointer, and call the new site's **IUnknown::AddRef**.

```
HRESULT SetSite(
    IUnknown * pUnkSite      // Pointer to IUnknown of the site managing this object
);
```

Parameter

pUnkSite

[in] The interface pointer of the site managing this object. If NULL, the object should call **IUnknown::Release** on any existing site at which point the object no longer knows its site.

Return Value

S_OK

Returned in all circumstances.

Remarks

E_NOTIMPL is not allowed—without implementation of the **SetSite** method, the **IObjectWithSite** interface is unnecessary.

See Also

IObjectWithSite::GetSite

IOleCommandTarget

The **IOleCommandTarget** interface defines a simple and extensible mechanism whereby objects and their containers can dispatch commands to each other. For example, an object's toolbars may contain buttons for commands such as Print, Print Preview, Save, New, and Zoom. Normal in-place activation guidelines recommend that you remove or disable such buttons because no efficient, standard mechanism has been available to dispatch them to the container. Similarly, a container has heretofore had no efficient means to send commands such as Print, Page Setup, and Properties to an in-place active object. Such simple command routing could have been handled through existing OLE Automation standards and the **IDispatch** interface, but the overhead with **IDispatch** is more than is required in the case of document objects. The **IOleCommandTarget** interface provides a simpler means to achieve the same ends.

Available commands are defined by integer identifiers in a group. The group is identified itself with a GUID. The interface allows a caller both to query for support of one or more commands within a group and to issue an order to the object to execute a particular command.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces
AddRef	Increments reference count
Release	Decrements reference count

IOleCommandTarget Methods	Description
QueryStatus	Queries object for status of commands
Exec	Execute a command

IOleCommandTarget::Exec

Executes a specified command or displays help for a command.

```
HRESULT Exec(
    const GUID *pguidCmdGroup,      // Pointer to command group
    DWORD nCmdID,                   // Identifier of command to execute
    DWORD nCmdExecOpt,              // Options for executing the command
    VARIANTARG *pvaIn,              // Input arguments
    VARIANTARG *pvaOut              // Pointer to output return values
  );
```

Parameters

pguidCmdGroup
[unique][in] Unique identifier of the command group; can be NULL to specify the standard group. The command passed in *nCmdID* must belong to this group.

nCmdID
[in] The command to execute, which must be in the group specified with *pguidCmdGroup.*

nCmdExecOpt
[in] One or more values from the OLECMDEXECOPT enumeration describing how the object should execute the command.

pvaIn
[unique][in] Pointer to a VARIANTARG containing input arguments. Can be NULL.

pvaOut
[unique][in,out] Pointer to a VARIANTARG to receive the output return values. Can be NULL.

Return Values

S_OK
The command was executed successfully.

E_UNEXPECTED
An unexpected error occurred.

E_FAIL
An error occurred

OLECMDERR_E_UNKNOWNGROUP
pguidCmdGroup is non-NULL but does not specify a recognized command group.

OLECMDERR_E_NOTSUPPORTED
The nCmdID argument is not recognized as a valid command in the group identified with pguidCmdGroup.

OLECMDERR_DISABLED
The command identified with nCmdID is currently disabled and cannot be executed.

OLECMDERR_NOHELP
The caller has asked for help on the command identified by nCmdID but no help is available.

OLECMDERR_CANCELED
The user canceled the execution of the action.

Remarks

The list of *in* and *out* arguments of a command and how they are packaged is unique to each command; such information should be documented with the specification of the command group (see

the Zoom command later in this section). In the absence of any specific information the command is assumed to take no arguments and have no return value.

Notes to Callers

The *pguidCmdGroup* and *nCmdID* arguments together uniquely identify the command to invoke. The *nCmdExecOpt* parameter specifies the exact action to take (see the OLECMDEXECOPT enumeration for more details).

Most of the commands neither take arguments nor return values. For such commands, the caller can pass NULL values for *pvaIn* and *pvaOut*. For commands that expect one or more input values, the caller can declare and initialize a VARIANTARG variable and pass a pointer to that variable in *pvaIn.* If the input to the command is a single value, the argument can be stored directly in the VARIANTARG and passed to the function. If the command expects multiple arguments, those arguments must be packaged appropriately within the VARIANTARG, using one of the supported types (such as **IDispatch***,* SAFEARRAY, etc.).

If a command returns one or more arguments, the caller is expected to declare a VARIANTARG, initialize it to VT_EMPTY, and pass its address in *pvaOut.* If the command returns a single value then the object can store that value directly in *pvaOut.* If the command has multiple output values then it will package those in some way appropriate for the VARIANTARG.

Because *pvaIn* and *pvOut* are both caller-allocated, stack variables are permitted for both the caller and the object receiving the call. For commands that take zero or one argument on input and return zero or one value, no additional memory allocation is necessary. Most of the types supported by VARIANTARG do not require memory allocation. Exceptions include SAFEARRAY and BSTR. For a complete list, see OLE documentation in the Win32 SDK.

Notes to Implementers

A command target must implement this function; therefore E_NOTIMPL is not a valid return value.

See Also

OLECMDEXECOPT

IOleCommandTarget::QueryStatus

Queries the object for the status of one or more commands generated by user interface events.

```
[input_sync] HRESULT QueryStatus(
    const GUID *pguidCmdGroup,      // Pointer to command group
    ULONG cCmds,                    // Number of commands in prgCmds array
    OLECMD *prgCmds,                // Array of OLECMD structures
    OLECMDTEXT *pCmdText            // Pointer to structure containing name or status of command
  );
```

Parameters

pguidCmdGroup
[unique][in] Unique identifier of the command group; can be NULL to specify the standard group. All the commands that are passed in the p*rgCmds* array must belong to the group specified by *pguidCmdGroup*.

cCmds
[in] The number of commands in the *prgCmds* array.

prgCmds
[in,out][size_is(cCmds)] A caller-allocated array of OLECMD structures, where the object receiving the call fills the *cmdID* field of each command with bits taken from the OLECMDF enumeration, specifying the status of each command.

pCmdText
[unique][in,out] Pointer to the structure in which to return name and/or status information. Can be NULL to indicate that the caller is not interested in such information.

Return Values

S_OK
The command status as any optional strings were returned successfully.

E_POINTER
The *prgCmds* argument is NULL.

E_UNEXPECTED
An unexpected error occurred.

E_FAIL
An error occurred

OLECMDERR_E_UNKNOWNGROUP
The *pguidCmdGroup* parameter is non-NULL but does not specify a recognized command group.

Remarks

Callers use **IOleCommandTarget::QueryStatus** to determine which commands are supported by a target object. The caller can then disable unavailable commands that would otherwise be routed to the object. The caller can also use this method to get the name or status of a single command.

Notes to Callers

The caller passes an array of OLECMD structures in *prgCmds* that describe the commands of interest from the group specified in *pguidCmdGroup*, where each structure's *cmdID* is set to a command identifier and the *cmdf* field is set to zero.

Notes to Implementers

The object receiving the call the fills the *cmdf* field for each command with bits taken from the OLECMDF enumeration to describe the status of each command.

The called object should first mark the command as described above. Then, if the command is supported (OLECMDF_SUPPORTED) the object should check the OLECMDTEXTF flags in the OLECMDTEXT structure. If the OLECMDFTEXF_NAME flag is specified, then the object should copy the localized name of the command (for example, “Open”, “Copy”, etc.) into the *rgwz* field of **OLECMDTEXT**, paying attention to the size specified by the *cwBuf* field in that same structure.

If the caller specifies OLECMDFTEXTF_STATUS, the object should copy a localized status string for the command into the *rgwz* field. The status string is typically contextual and depends on the state of the command—enabled/disabled, for example. If the buffer is not big enough then the object should zero terminate the buffer. Whether the buffer is big enough or not the object must return the total actual size of the string(s), that it attempted to copy, via the *cwActual* field.

If the command array contains more than one command, the textual information should be returned for the first command in the command array that the object supports. Typically, this functionality is used to show the status text of a command. Note that the caller can use a either stack or a global variable for *rgwz* because memory for this parameter is not dynamically allocated.

Because **QueryStatus** is defined with the [input_sync] attribute, the implementing object cannot yield or make another non input_sync RPC call while executing it.

A command target must implement this function. Therefore, E_NOTIMPL is not a valid return value.

See Also

OLECMD, OLECMDF, OLECMDTEXT

IOleDocument

The **IOleDocument** interface enables a document object to communicate to prospective containers its ability to create views of its data. This interface also enables a document object to enumerate its views and to provide containers with miscellaneous information about itself, such as whether it supports multiple views or complex rectangles.

When to Implement

An OLE document that is to be activated as a document object at the very least must implement this interface. If an application exposes more than one type of document object, each type must implement **IOleDocument**. A document's view frame provider, which

When to Use

A container of document objects calls the methods of this interface to ask a document object to create views of itself, enumerate the views it supports, or provide miscellaneous information about its capabilities.

Methods in VTable Order

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

IOleDocument Methods	Description
CreateView	Creates a new view object.
GetDocMiscStatus	Returns status bits about the document object.
EnumViews	Enumerates views supported by the document object.

See Also

IOleCommandTarget, **IOleContinueCallback**, IOleDocumentSite, IOleDocumentView, IPrint

IOleDocument::CreateView

Creates a document view object in the caller's process and returns a pointer to that object's **IOleDocumentView** interface.

```
HRESULT CreateView(
    IOleInPlaceSite * pIPSite,        // Pointer to container's view-site object
    IStream * pstm,                   // Pointer to a stream object
    DWORD dwReserved,                 // Reserved for future use
    IOleDocumentView * ppView         // Indirect pointer to the new document view object
  );
```

Parameters

pIPSite

[in] Pointer to the view-site object to be associated with the new document view object. May be NULL, as when creating a new, uninitialized document object, in which case the caller must initialize the view with a subsequent call to IOleDocumentView::SetInPlaceSite.

pstm

[in] Pointer to a stream containing data from which the new document view object should initialize itself. Normally will be NULL when the specified view is for a new, uninitialized document object. If NULL, the document object initializes the new document view object with a default state.

dwReserved

[in] Reserved for future use. In the future this parameter could be used to specify the type of view that needs to be created. Currently there are no defined values for this argument.

ppView

[out] Indirect pointer to the new document view object. If **CreateView** succeeds, the caller is responsible for calling **Release** through this pointer when the view object is no longer needed.

Return Values

S_OK

The view was created successfully.

E_POINTER

The address in *ppView* is NULL.

E_OUTOFMEMORY

Out of memory.

E_UNEXPECTED

An unexpected error occurred.

E_FAIL

An unspecified error occurred.

Remarks

A document-object container's document site calls **IOleDocumentView::CreateView** to instruct a document object to create a new view of itself in the container's process, either from scratch or by using the contents of an existing stream.

Calling **CreateView** does not cause the new view to display itself. To do so, the view must wait for calls to either IOleDocumentView::Show or IOleDocumentView::UIActivate.

Notes to Callers

The call to **CreateView** normally occurs in the container's implementation of

IOleDocumentSite::ActivateMe**.** If a document object passes an **IOleDocumentView** pointer in its call to **ActivateMe**, the container has no need to call **CreateView** and instead can call **IOleDocumentView::SetInPlaceSite**, followed by **AddRef**. If a document object returns a NULL view pointer in its call to **ActivateMe**, then the container calls **CreateView** to retrieve a view pointer.

The container has the option of passing a pointer to a stream containing data to be used to initialize the requested view. If the container passes a NULL stream pointer, then the document object will initialize the new view using its default settings.

Notes to Implementers

This method must be completely implemented for any document object. Therefore E_NOTIMPL is not an acceptable return value.

If *pIPSite* is non-NULL, then the document object should pass the pointer to the new view through IOleDocumentView::SetInPlaceSite. If *pIPSite* is NULL, the caller is responsible for making this same call. In addition, if *pstm* is non-NULL, then the object should initialize the view object by passing *pstm* to IOleDocumentView::ApplyViewState.

As with all new interface pointers, **CreateView** should call **AddRef** on the pointer in **ppView* before returning. The caller is responsible for calling **Release** through this pointer when it is no longer needed.

See Also

IOleDocumentSite::ActivateMe, IOleDocumentView::ApplyViewState, IOleDocumentView::SetInPlaceSite, IOleDocumentView::Show, IOleDocumentView::UIActivate

IOleDocument::EnumViews

Creates an object that enumerates the views supported by a document object or, if only one view is supported, returns a pointer to that view.

```
HRESULT EnumViews(
    IEnumOleDocumentViews ** ppEnum,      // On return, pointer to interface pointer to an enumerator
    IOleDocumentView ** ppView            // On return, pointer to interface pointer to a single view
  );
```

Parameters

ppEnum
[out] Indirect pointer to the enumerator's IEnumOleDocumentViews interface.

ppView
[out] Indirect pointer to a single view object's IOleDocumentView interface.

Return Value

S_OK
If the object supports multiple views, then **ppEnum* contains the enumerator pointer. Otherwise **ppEnum* is NULL and **ppView* contains the interface pointer to the single view.

E_POINTER
The address in *ppEnum* or *ppView* is invalid. The caller must pass pointers for both arguments.

E_OUTOFMEMORY
The enumerator could not be created because there is insufficient memory.

Remarks

If a document object supports multiple views of its data, it must also implement **IEnumOleDocumentViews** and return that interface's pointer in the out-parameter * *ppEnum*. Using this pointer, the container can enumerate the views supported by the document object.

If the document object supports only a single view, **IOleDocument::EnumViews** returns that view's **IOleDocumentView** pointer in the out parameter **ppView*.

Notes to Callers

Call this method to determine if a document object supports more than one view of its data. If it does, the caller can use the pointer returned in **ppEnum* to specify which view to activate. When finished with the pointer, the caller must free it by calling **Release**.

Notes to Implementers

This method must be completely implemented on all document objects; E_NOTIMPL is not an acceptable return value. The way in which this method should be implemented depends on whether the document object in question supports multiple views or only a single view.

See Also

IEnumOleDocumentViews, IOleDocumentView

IOleDocument::GetDocMiscStatus

Returns information about whether a document object supports behaviors specified in the DOCMISC enumerator. Values from this enumerator are also stored in the registry as the value of the “DocObject” key.

```
HRESULT GetDocMiscStatus(
    DWORD *pdwStatus      // Pointer to information about the document object
  );
```

Parameters

pdwStatus

[out] On return, a pointer to the information on supported behaviors; may be NULL. Values written to this pointer are taken from the **DOCMISC** enumeration.

Return Values

S_OK

The status bits were returned successfully

E_POINTER

The address in *pdwStatus* is NULL.

Remarks

This method provides a way for containers to ascertain whether a document object supports multiple views, complex rectangles, opening in a pop-up window, or file read/write.

Notes to Callers

By calling this method prior to activating a document object, containers can take whatever steps are necessary to support, or otherwise accommodate, the specified behaviors.

Notes to Implementers

This function must be completely implemented in any document object even if a zero is returned; therefore E_NOTIMPL is not an acceptable return value. Normally, the returned **DOCMISC** value should be hard-coded for performance.

See Also

DOCMISC

IOleDocumentSite

The **IOleDocumentSite** interface enables a document that has been implemented as a document object to bypass the normal activation sequence for in-place-active objects and to directly instruct its client site to activate it as a document object. A client site with this ability is called a *document site*.

For each document object to be hosted, a container must provide a corresponding document site, which is an OLE Documents client site that, in addition to implementing **IOleClientSite** and **IAdviseSink**, implements IOleDocumentSite. Each document site implements a separate document view site for each view of a document to be activated. The document view site implements **IOleInPlaceSite** and, optionally, IContinueCallback.

When to Implement

Implement **IOleDocumentSite** on each and every client site that is to host a document object. In addition to this interface, a document site must also implement **IOleClientSite**, **IOleInPlaceSite**, and **IAdviseSink**. **IContinueCallback** is optional.

When to Use

A document object calls **IOleDocumentSite::ActivateMe** to ask its document site to activate it, typically in response to the container's call to **IOleObject::DoVerb**.

Methods in VTable Order

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

IOleDocumentSite Methods	Description
ActivateMe	Activates an OLE Document Object

IOleDocumentSite::ActivateMe

Asks a document site to activate the document making the call as a document object rather than an in-place-active object and, optionally, specifies which view of the object document to activate.

```
HRESULT ActivateMe(
    IOleDocumentView * pViewToActivate      // Pointer to the view to be activated
  );
```

Parameters

pViewToActivate

[in] Pointer to the document view to be used in activating the document object. Can be NULL, in which case the container should call IOleDocument::CreateView to obtain a document view pointer.

Return Value

S_OK

The container activated the view successfully.

E_OUTOFMEMORY

pViewToActivate is NULL and the container's call to **IOleDocument::CreateView** failed with E_OUTOFMEMORY.

E_FAIL

Another error occurred in either view creation or activation.

Remarks

When a container calls **IOleObject::DoVerb** to request a document to activate, a document object bypasses the usual in-place activation sequence by calling **IOleDocumentSite::ActivateMe**.

When calling **IOleObject::DoVerb** on a document object, containers will usually find OLEIVERB_SHOW to be the most appropriate activation verb. Other allowable verbs include OLEIVERB_PRIMARY and OLEIVERB_UIACTIVATE. OLEIVERB_OPEN is discouraged because it implies opening an embedded object in a separate window, which is contrary to the intent of document object activation.

Notes to Callers

Only OLE document objects should call this method. A normal in-place active OLE document should respond to a container's call to **IOleObject::DoVerb** by calling **IOleInPlaceSite.**

A document object should initiate its activation by calling **IOleDocumentSite::ActivateMe**. If the container does not implement **IOleDocumentSite**, then the document should default to the normal in-place activation sequence.

A document object that supports more than one view of its data can specify which view to activate by passing a pointer to that view's IOleDocumentView interface in *pViewToActivate*.

However the **IOleDocumentView** pointer is obtained, the container should release the pointer when it is no longer needed.

Notes to Implementers

This function must be completely implemented in a document object container; E_NOTIMPL is not an acceptable return value.

If a document object passes an **IOleDocumentView** pointer in *pViewToActivate*, the container's implementation of **ActivateMe** should call IOleDocumentView::SetInPlaceSite and pass a

pointer to its **IOleInPlaceSite** interface back to the document. If the container is holding onto the **IOleDocumentView** pointer, which will normally be the case, it should follow the call to **SetInPlaceSite** with a call to **AddRef**.

If a document object sets the value of *pViewToActivate* to NULL, the container can obtain a pointer to a document view by querying the document for **IOleDocument**, then calling IOleDocument::CreateView and passing its **IOleInPlaceSite** pointer.

See Also

IOleClientSite, IOleDocument::CreateView, IOleDocumentView::SetInPlaceSite, **IOleInPlaceSite**, **IOleObject::DoVerb**

IOleDocumentView

The **IOleDocumentView** interface enables a container to communicate with each view supported by a document object. A document object that supports multiple views of its data represents each view as a separate object. Each document view object implements **IOleDocumentView,** along with **IOleInPlaceObject***,* **IOleInPlaceActiveObject***,* and optional interfaces such as **IPrint** and **IOleCommandTarget**. A document object that supports only a single view does not require that view to be implemented as a separate object. Instead, both document and view can be implemented as a single class.

The **IOleDocumentView** interface provides all the necessary methods for a container to manipulate, manage, and activate a document view object.

The **SetInPlaceSite** method enables a container to hand a document object an **IOleInPlaceSite** pointer to its view site in the container. The method **GetInPlaceSite** enables a container to retrieve a document object's **IOleInPlaceSite** pointer.

The **GetDocument** method provides a container with access to the **IUnknown** pointer of the document object containing the view object to which the call is made.

The **SetRect** and **GetRect** methods manage the simple rectangle that a view will occupy in the container. **SetRectComplex** enables a container to specify not only the simple rectangle but also the spaces that should be occupied by the view's scrollbars and size box.

The **Show***,* **UIActivate**, and **Open** methods enable a container to manipulate a document view object's visual state. **Show** instructs a document view object to activate or deactivate itself in-place. When a view object is active, **UIActivate** instructs it to activate or deactivate its user interface elements, such as menus, toolbars, and accelerators. **IOleDocumentView::Show** provides document objects with the same functionality that the **IOleInPlaceObject** methods **InPlaceActivate** and **InPlaceDeactivate** provide to normal in-place-active objects; **IOleDocumentView::UIActivate**, the same functionality as **UIActivate** and **UIDeactivate**.

The **Open** method activates a document object in a separate window, as is the case with an embedded OLE Document that does not support in-place activation.

The **Close** method instructs a document view object to deactivate itself and release its document view site pointer. If the view is active in a separate window, the document view object should also close that window

The **SaveViewState** method enables a container to save a view's internal state to a stream. The **ApplyViewState** method enables a container to reload a previously saved view state.

The **Clone** method enables a container to create a duplicate of the current view object.

When to Implement

All document objects must implement **IOleDocumentView** for each and every view they support. If a document object supports multiple views, each view must be implemented as a separate class object. If a document object supports only a single view, the document object and its view can both be implemented as a single class.

When to Use

A container calls the methods of this interface to activate, deactivate, close, and generally communicate with a document view object.

Methods in VTable Order

IUnknown Methods	Description

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

IOleDocumentView Methods	**Description**
SetInPlaceSite	Associates a container site with this view.
GetInPlaceSite	Retrieves the last site passed in **SetInPlaceSite**.
GetDocument	Returns the **IUnknown** pointer of the document object that owns this view.
SetRect	Sets the coordinates of the view port.
GetRect	Retrieves the coordinates last passed in **SetRect**
SetRectComplex	Sets the coordinates of the view port, scroll bars, and size box.
Show	In-place activates or deactivates a view.
UIActivate	In-place activates or deactivates a view's user-interface elements.
Open	Displays view in a separate pop-up window
Close	Instructs view to close
SaveViewState	Saves view state into stream
ApplyViewState	Initializes view with view state previously saved in call to **SaveViewState**.
Clone	Creates a duplicate view object.

See Also

TBD

IOleDocumentView::ApplyViewState

Instructs a view to reinitialize itself according to the data in a stream that was previously written by means of a call to IOleDocumentView::SaveViewState.

```
HRESULT ApplyViewState(
    IStream *pstm       // Pointer to stream
  );
```

Parameters

pstm
[in] Pointer to the stream from which the view should load its state.

Return Values

S_OK
The view successfully loaded its state from the specified stream.

E_POINTER
The value in *pstm* is NULL.

E_NOTIMPL
This view has no meaningful state to load. This error should be rare because most views will have at least some state information worth loading.

Remarks

Typically, this function is called after an existing view has been created in the container but before that view has been displayed. It is the responsibility of the view to validate the data in the view stream; the container does not attempt to interpret view-state stream data.

The semantics of **ApplyViewState** are encompassed in the *pstm* argument of IOleDocument::CreateView.

See Also

IOleDocument::CreateView, IOleDocumentView_SaveViewState

IOleDocumentView::Clone

Creates a duplicate view object with an identical internal state to the current view.

```
HRESULT Clone(
    IOleInPlaceSite *pIPSiteNew          // Pointer to view site
    IOleDocumentView **ppViewNew         // On return, pointer to interface pointer to view object
);
```

Parameters

pIPSiteNew

[in] Pointer to the in-place site in which the view to be cloned will be activated. The view being cloned should pass this pointer to the cloned view's IOleDocumentView::SetInPlaceSite method. This pointer can be NULL, in which case the caller is responsible for calling **SetInPlaceSite** on this new view directly.*ppViewNew*

[out] Indirect pointer to the new view object's IOleDocumentView interface. The caller is responsible for this pointer any must release it when it is no longer needed.

Return Values

S_OK

The view was successfully cloned.

E_POINTER

The value in *ppViewNew* is NULL.]

E_NOTIMPL

The view object does not implement this interface.

E_FAIL

The document object does not implement this interface.

This is useful for creating a new view with a different view port and view site but with the same view context as the view being cloned. Typically this will be used to implement the "Window-New window" functionality.

Remarks

This method is useful for creating a new view with a different view port and view site but with the same view context as the view being cloned. Typically, containers hosting an MDI application will call this method to provide "Window/New window" capability.

See Also

IOleDocumentView, IOleDocumentView::SetInPlaceSite

IOleDocumentView::Close

Asks a document view to close and release its **IOleInPlaceSite** pointer.

HRESULT Close(
  DWORD *dwReserved* // Reserved. Must be zero
);

Return Values

S_OK
  The view successfully closed itself.

Remarks

The container must call this method before it wants to delete the view (that is, release its last reference to the view). In general, implementaion of this method will call IOleDocumentView::Show(FALSE) to hide the view if it is not already, then call IOleDocumentView::SetInPlaceSite(NULL) to deactivate itself and release the view site pointer.

Because **Close** is called when a container is going to completely shut down a view, this method must be implemented and has no reason to fail.

See Also

IOleDocumentView::SetInPlaceSite, IOleDocumentView::Show

IOleDocumentView::GetDocument

Returns the **IUnknown** interface pointer of the document object that owns this view.

HRESULT GetDocument(
  IUnknown ***ppunk** // On return**,** pointer to the **IUnknown** pointer of the document object that owns this view
);

Parameters

ppunk
  [out] Pointer to the **IUnknown** pointer of the document object.

Return Value

S_OK
  The document object's interface pointer was successfully returned. This is the only valid return value for this function.

Remarks

The caller is responsible incrementing the reference count on the interface pointer returned by this method. The caller must call **Release** through this pointer when it is no longer needed.

Because a document owning the view must always exist, this method will always succeed. Before returning, this method should call **AddRef** on the pointer stored in **ppunk*.

See Also

IUnknown

IOleDocumentView::GetInPlaceSite

Retrieves the most recent view site associated with this view of the document.

HRESULT GetInPlaceSite(
  IOleInPlaceSite ***ppIPSite** // On return**,** Indirect pointer to the document's view site
);

Parameters

ppIPSite
  [out] Pointer to the view site's **IOleInPlaceSite** interface pointer.

Return Value

S_OK
  The interface pointer was returned successfully.

E_FAIL
  Another error occurred.

Remarks

Returns the most recent **IOleInPlaceSite** pointer passed in IOleDocumentView::SetInPlaceSite, or NULL if **SetInPlaceSite** has not yet been called. The view will call **AddRef** on this pointer before returning it, thus the caller must later call **Release**.

Notes to Callers

The caller is responsible incrementing the reference count on the interface pointer returned by this method. The caller must call **Release** through this pointer when it is no longer needed.

Notes to Implementers

A document view must implement this method completely. Therefore E_NOTIMPL is not an acceptable return value.

See Also

IOleDocumentView::SetInPlaceSite, **IOleInPlaceSite**

IOleDocumentView::GetRect

Returns the rectangular coordinates of the view port in which the view is to be activated.

```
HRESULT GetRect(
    LPRECT prcView      // Pointer to the current view coordinates
);
```

Parameters

prcView

[out] Pointer to a RECT structure to which the document view object will write the coordinates of its current view port.

Return Value

S_OK

The view was successfully resized to the rectangle.

E_UNEXPECTED

This view has not yet seen a call to IOleDocumentView::SetRect or IOleDocumentView::SetRectComplex and therefore has no rectangle to return.

Remarks

For an SDI application, the view port is the client area of the frame window minus the space allocated for tool bars, status bar, and such. For an MDI window, the view port is the client area of the MDI document window minus any other frame-level user-interface elements.

The view port coordinates returned by this method are those set in the last previous call to either **IOleDocumentView::SetRect** or **IOleDocumentView::SetRectComplex**.

A document view must implement this method completely. Therefore E_NOTIMPL is not an acceptable return value.

See Also

IOleDocumentView::SetRect, IOleDocumentView::SetRectComplex

IOleDocumentView::Open

Asks a document view to display itself in a separate pop-up window with semantics equivalent to **IOleObject::DoVerb**(OLEIVERB_OPEN).

HRESULT Open(Void);

Return Values

S_OK
The view successfully created its separate window.

E_OUTOFMEMORY
There was not enough memory to activate the view in a separate window.

E_FAIL
Some other error occurred that prevented success.

E_NOTIMPL
The document object that owns this view does not support separate window activation.

E_UNEXPECTED
This method was called before a call to IOleDocumentView::SetInPlaceSite.

Remarks

A user viewing a document object in a container application such as a browser or “binder” may want to see two or more views or documents at once. Because the browser displays only one view at a time, the container needs a way to instruct the other views or documents to display themselves, as required, in separate windows. The IOleDocumentView::Open interface provides that way.

Notes to Callers

A successful call to **IOleDocumentView::Open** should be followed by a call to IOleDocumentView::Show(FALSE) to hide the window or **IOleDocumentView::Show**(TRUE) to show the window and bring it to the foreground. While the view is active in its separate window, a container can show or hide the window as many times as it may require.

Notes to Implementers

A document object that does not support activation in a separate window indicates as much by setting DOCMISC_CANTOPENEDIT in its DOCMISC enumeration and returning E_NOTIMPL to containers that nevertheless call this method in an attempt to open it. Objects that have a limited interface for activation purposes should be sure to set DOCMISC_CANTOPENEDIT.

Otherwise, implementation generally consists of calling the view’s own interface, in response to which the view shows its separate pop-up window and brings that window to the foreground.

When the separate window is no longer needed, the container calls IOleDocumentView::Close, whereupon the view releases its site pointer to the separate window and destroys the window. Contrary to the normal in-place deactivation sequence for OLE Documents, however, a document view continues to hold the **IOleInPlaceSite**, pointer that it obtained from the container’s call to **IOleDocumentView::SetInPlaceSite**. This pointer is only released when the view's container calls **IOleDocumentView::SetInPlaceSite** and sets *pIPSite* to NULL or calls **IOleDocumentView::CloseView**.

When a user closes a view’s separate window, the view should not shut itself down. Instead, it should call **IOleInPlaceSite::OnInPlaceActivate**. The view site then decides whether to call IOleDocumentView::UIActivate(FALSE) immediately or later. In this way, a document view displayed in a separate window remains available for activation in the container's own window.

See Also

IOleDocumentView::Close, IOleDocumentView::Show, IOleDocumentView::UIActivate, **IOleInPlaceObject::InPlaceDeactivate**, **IOleInPlaceSite**, **IOleInPlaceSite::OnInPlaceActivate**

IOleDocumentView::SaveViewState

Saves the view's state into the specified stream.

```
HRESULT SaveViewState(
    IStream *pstm       // Pointer to stream
  );
```

Parameters

pstm
[in] Pointer to stream in which the view is to save its state.

Return Values

S_OK
The view successfully saved its state to the specified stream.

E_POINTER
The value in *pstm* is NULL.

E_NOTIMPL
This view has no meaningful state to save. This error should be rare because most views have at least some state information worth saving.

Remarks

The view's state includes properties such as the view type, zoom factor, and insertion point. The container typically calls this function before deactivating the view. The stream can then later be used to reinitialize a view of the same document to this saved state through IOleDocumentView_ApplyViewState.

According to the rules governing **IPersistStream**, a view must write its CLSID as the first element in the stream. Any cross-platform file format compatibility issues that apply to the document's storage representation also apply to this context.

See Also

IOleDocumentView_ApplyViewState, **IPersistStream**

IOleDocumentView::SetInPlaceSite

Associates a container's document view site with a document's document view object.

HRESULT SetInPlaceSite(
  IOleInPlaceSite * *pIPSite* // Pointer to document view site's **IOleInPlaceSite** interface
);

Parameters

pIPSite
  [in] Pointer to the document view site's **IOleInPlaceSite** interface. Can be NULL, in which case the document view object loses all association with the container.

Return Value

S_OK
  A document view site was successfully associated (or disassociated if *pIPSite* is NULL) with a document view object.

E_FAIL
  An unspecified error occurred.

Remarks

As part of activating a document object, a container must hand the object a pointer to the container's implementation of **IOleInPlaceSite**. This pointer designates the document view site that is to be associated with the particular view of the document to be activated.

A container normally passes this pointer in response to a document's request to be activated. A document makes such a request by calling IOleDocumentSite::ActivateMe and handing the container a pointer to the view to be activated. The container, in turn, uses this pointer to call **IOleDocumentView::SetInPlaceSite**.

Notes to Callers

If the container is requesting creation and activation of a new instance of a document object, rather than merely the activation of a loaded instance of a document object, the semantics of **SetInPlaceSite** are encompassed in the *pIPSite* argument of IOleDocument::CreateView**.** Therefore, an explicit call to **SetInPlaceSite** is unnecessary.

Notes to Implementers

If this method is called on a view that already has an associated view site, the view must do some housekeeping in preparation for activating itself in the new site. First, the view must deactivate itself in the current site. Second, the view must release its pointer to that site. Third, if the new **IOleInPlaceSite** pointer is not NULL, the view should both save the pointer and call **AddRef** on it. Having completed these steps, the view should then wait for the container to tell it when to activate itself in the new view site.

A document view must implement this method completely. Therefore E_NOTIMPL is not an acceptable return value.

See Also

IOleDocumentView::GetInPlaceSite

IOleDocumentView::SetRect

Sets the rectangular coordinates of the view port in which a view is to be activated.

[input_sync] HRESULT SetRect(
  LPRECT *prcView* // Pointer to a RECT structure
);

Parameters

prcView
  [in] Pointer to a RECT structure containing the coordinates of the view port.

Return Value

S_OK
  The view was successfully resized.

E_FAIL
  Some other critical error occurred that prevented the view from being resized.

Remarks

For an SDI application, the view port is the client area of the frame window minus the space allocated for tool bars, status bar, and such. For an MDI window, the view port is the client area of the MDI document window minus any other frame-level user-interface elements.

Notes to Callers

Calling **IOleDocumentView::SetRect** (or SetRectComplex) is part of the normal activation sequece for document objects, usually following a call to IOleDocumentView::UIActivate and preceding a call to IOleDocumentView::Show. In addition, whenever the window used to display a document object is resized, the container should call **SetRect** (or **SetRectComplex**) to tell the document object to resize itself to fit the new window dimensions.

Notes to Implementers

The coordinates of the view port are within the coordinates of the view window, which is obtained through **IOleInPlaceSite::GetWindow**. The view must resize itself to fit the new coordinates.

This method is defined with the [input_sync] attribute, which means that the implementing object cannot yield or make another, non input_sync RPC call while executing this method.

A document view must implement this method completely. Therefore E_NOTIMPL is not an acceptable return value.

See Also

IOleInPlaceSite::GetWindow

IOleDocumentView::SetRectComplex

Sets the rectangular coordinates of the view port, horizontal and vertical scroll bars, and the size box.

```
[input_sync] HRESULT SetRectComplex(
    LPRECT prcView,        // Pointer to coordinates of view port
    LPRECT prcHScroll,     // Pointer to coordinates of horizontal scroll bar
    LPRECT prcVScroll,     // Pointer to coordinates of vertical scroll bar
    LPRECT prcSizeBox      // Pointer to coordinates of size box
  );
```

Parameters

prcView
[in] Pointer to a RECT structure containing the coordinates of the view port.

prcHScroll
[in] Pointer to a RECT structure containing the coordinates of the horizontal scroll bar.

prcVScroll
[in] Pointer to a RECT structure containing the coordinates of the vertical scroll bar.

prcSizeBox
[in] Pointer to a RECT structure containing the coordinates of the size box.

Return Value

S_OK
The view was successfully resized to the rectangle.

E_NOTIMPL
The document object that owns this view does not support complex rectangles.

E_FAIL
Some other critical error occurred that prevented resizing the view or placing the scrollbars and size box.

Remarks

View frames that support a workbook metaphor, in which a single document comprises multiple sheets or pages, typically call this method to set the coordinates to be used in common by all the sheets or pages.

Notes to Callers

Calling **IOleDocumentView::SetRectComplex** is part of the normal activation sequece for document objects that support complex rectangles, usually following a call to IOleDocumentView::UIActivate and preceding a call to IOleDocumentView::Show. In addition, whenever the window used to display a document object is resized, the container should call **SetRectComplex** (or IOleDocumentView::SetRect) to tell the document object to resize itself to fit the new window dimensions.

Notes to Implementers

Document objects that support complex rectangles mark themselves with DOCMISC_SUPPORTCOMPLEXRECTANGLES, as described in DOCMISC and IOleDocument::GetDocMiscStatus. Document objects that do not support this method can return E_NOTIMPL.

Upon receiving a call to this method, a view should resize itself to fit the coordinates specified in

prcView and fit its scrollbars and size box to the areas described in *prcHScroll, prcVScroll,* and *prcSizeBox*.

This method is defined with the [input_sync] attribute, which means that the implementing object cannot yield or make another, non input_sync RPC call while executing this method.

See Also

DOCMISC , IOleDocument::GetDocMiscStatus

IOleDocumentView::Show

Instructs a view to in-place activate or in-place deactivate itself.

HRESULT Show(
  BOOL *fShow* // Boolean value specifying whether to activate or deactivate the view
);

Parameters

fShow
  [in] TRUE instructs the view to show itself; FALSE instructs the view to hide itself.

Return Value

S_OK
  The view was successfully shown or hidden.

E_OUTOFMEMORY
  There was not enough memory to activate or hide the view.

E_FAIL
  Some other critical error occurred that prevented activation or hiding.

E_UNEXPECTED
  This method was called before a call to IOleDocumentView::SetInPlaceSite.

Remarks

Calling **IOleDocumentView::Show** is the last step in the activation sequence because before showing itself a document object must know exactly what space it occupies and have all its tools available.

Notes to Callers

A call to this method for the purpose of activating a view should follow calls to **IOleDocumentView::SetInPlaceSite**, IOleDocumentView::UIActivate, and IOleDocumentView::SetRect (or SetRectComplex).

Notes to Implementers

Implementations of this method should embody the following pseudo-code:

```
if (fShow)
    {
    In-place activate the view but do not UI activate it.
    Show the view window.
    }
else
    {
    Call IOleDocumentView::UIActivate(FALSE) on this view
    Hide the view window
    }
```

All views of a document object must at least support the in-place activation mode, therefore E_NOTIMPL is not allowed as a return value.

See Also

IOleDocumentView::SetInPlaceSite, IOleDocumentView::SetRect, IOleDocumentView::SetRectComplex, IOleDocumentView::UIActivate

IOleDocumentView::UIActivate

Activates or deactivates a document view's user-interface elements, such as menus, toolbars, and accelerators.

```
HRESULT UIActivate(
    BOOL fUIActivate      // Boolean value specifying whether to activate or deactivate view
  );
```

Parameters

fUIActivate
[in] TRUE instructs the view to activate its user interface; FALSE instructs the view to deactivate its user interface.

Return Value

S_OK
The view's user interface was successfully activated or deactivated.

E_OUTOFMEMORY
There was not enough memory to activate the view's user interface.

E_FAIL
Some other critical error occurred that prevented activation or deactivation of the user interface.

E_UNEXPECTED
This method was called before a call to IOleDocumentView::SetInPlaceSite.

Remarks

Calling this method is part of the normal activation and deactivation sequences of a document object.

Notes to Callers

Calling this method before calling **IOleDocumentView::SetInPlaceSite** returns an error because the view must know where it is activating itself before it can proceed to do so. When calling **IOleDocumentView::UIActivate** as part of the activation sequence, the call should precede one to IOleDocumentView::SetRect (or SetRectComplex) because otherwise the container would send to the view dimensions that would be incorrect because they would not account for toolbar space.

When deactivating a view, the container should call IOleDocumentView::Show(FALSE), followed by IOleDocumentView::UIActivate(FALSE).

Notes to Implementers

Implementations of this method should embody the following pseudo-code:

```
if (fActivate)
    {
    UI activate the view (do menu merging, show frame level tools, process accelerators)
    Take focus, and bring the view window forward.
    }
else
    call IOleInPlaceObject::UIDeactivate on this view
```

In addition, the view may, and should, participate in extended Help menu merging.

All views of a document object must at least support the in-place activation mode. Therefore, E_NOTIMPL is not allowed as a return value.

See Also

IOleDocumentView::SetInPlaceSite, IOleDocumentView::SetRect, IOleDocumentView::SetRectComplex, IOleDocumentView::Show, **IOleInPlaceObject::UIDeactivate**

IParseDisplayName

The **IParseDisplayName** interface parses display names for custom moniker implementations.

When to Implement

Monikers that support their own namespace with custom requirements for parsing names implement this interface.

When to Use

MkParseDisplayName[Ex] calls this interface to parse display names for objects that provide custom moniker implementations.

Methods in Vtable Order

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

IParseDisplayName Method	Description
ParseDisplayName	Parses a display name returning a moniker corresponding to it.

IParseDisplayName::ParseDisplayName

Parses a display name and returns a moniker representing it.

HRESULT ParseDisplayName(

IBindCtx* *pbc*,	//Pointer to bind context
LPWSTR *szDisplayName*,	//Display name to be parsed
ULONG **pcchEaten*,	//Pointer to the number of characters of the display name successfully parsed
IMoniker ***** *ppmk*	//Indirect pointer to a moniker

);

Parameters

pbc
[in] Pointer to the bind context in which to accumulate bound objects.

szDisplayName
[in] Display name to be parsed.

pcchEaten
[out] Pointer to the number of characters of the display name that were consumed in this parse.

ppmk
[out] Indirect pointer to the resulting moniker.

Return Values

S_OK
The operation was successful.

MK_E_SYNTAX
Parsing failed because *szDisplayName* could only be partially resolved into a moniker. In this case, **pcchEaten* has the number of characters that were successfully parsed into a moniker prefix.

E_INVALIDARG
One or more parameters are invalid.

Remarks

In general, the maximal prefix of *szDisplayName* that is syntactically valid and that represents an object should be consumed.

The initial step of **MkParseDisplayName[Ex]** can retrieve this interface directly from an instance of the class identified with either “@ProgID” or “ProgID” notation. Later parsing steps can request this interface on an intermediate object.

The main loop of **MkParseDisplayName[Ex]** finds the next moniker piece by calling the **IMoniker**-equivalent function (**IMoniker::ParseDisplayName**) on the moniker-so-far that it holds on to, passing NULL in the *pmkToLeft* parameter. In case the moniker-so-far is a generic composite, this is forwarded by that composite onto its last piece, passing the prefix of the composite to the left of the piece in *pmkToLeft*.

Some moniker classes will be able to handle this parsing internally to themselves since they are designed to designate only certain kinds of objects. Others will need to bind to the object that they designate to accomplish the parsing process. As is usual, these objects should not be released by **IMoniker::ParseDisplayName** but instead should be transferred to the bind context (via **IBindCtx::RegisterObjectBound** or **IBindCtx::GetRunningObjectTable** followed by **IRunningObjectTable::Register**) for release at a later time.

If a syntax error occurs, then NULL should be returned through *ppmkOut* and MK_E_SYNTAX returned. In addition, the number of characters of the display name that were successfully parsed should be returned through *pcchEaten*.

See Also

MkParseDisplayNameEx

IPersistMemory

IPersistMemory operates exactly as **IPersistStreamInit**, except that it allows the caller to provide a fixed-size memory block (identified with a **void** *) as opposed to **IPersistStreamInit** which involves an arbitrarily expandable **IStream**.

The *cbSize* argument to the **Load** and **Save** methods indicate the amount of memory accessible through *pvMem*.

The **IsDirty**, **GetSizeMax**, and **InitNew** methods are semantically and syntactically identical to those in **IPersistStreamInit**. Only **Load** and **Save** differ.

Methods in Vtable Order

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

IPersist Method	Description
GetClassID	Returns the class identifier (CLSID) for the component object.

IPersistMemory Methods	Description
IsDirty	Checks the object for changes since it was last saved.
Load	Initializes an object from the memory block where it was previously saved.
Save	Saves an object into the specified memory block and indicates whether the object should reset its dirty flag.
GetSizeMax	Returns the size in bytes of the memory block needed to save the object.
InitNew	Initializes an object to a default state.

IPersistMemory::GetSizeMax

Returns the size in bytes of the memory block needed to save the object.

```
HRESULT GetSizeMax(
    ULARGE_INTEGER *pcbSize      // Pointer to size of memory needed to save object
  );
```

Parameter

pcbSize

[out]Points to a 64-bit unsigned integer value indicating the size in bytes of the memory needed to save this object.

Return Value

S_OK

The size was successfully returned.

Remarks

This method returns the size needed to save an object. You can call this method to determine the size and set the necessary buffers before calling the **IPersistMemory::Save** method.

Notes to Implementors

The **GetSizeMax** implementation must return a conservative estimate of the necessary size because the **IPersistMemory::Save** method uses a fixed size memory block.

See Also

IPersistMemory::Save

IPersistMemory::InitNew

Initializes the object to a default state. This method is called instead of **IPersistMemory::Load**.

HRESULT InitNew(void);

Return Values

S_OK
: The object successfully initialized itself.

E_NOTIMPL
: The object requires no default initialization. This error code is allowed because an object may choose to implement **IPersistMemory** simply for orthogonality or in anticipation of a future need for this method.

E_UNEXPECTED
: This method was called after the object was already initialized with **IPersistMemory::Load**. Only one initialization is allowed per instance.

E_OUTOFMEMORY
: There was not enough memory for the object to initialize itself.

Notes to Implementers

If the object has already been initialized with **Load**, then this method must return E_UNEXPECTED.

See Also

IPersistMemory::Load

IPersistMemory::IsDirty

Checks the object for changes since it was last saved.

HRESULT IsDirty(void);

Return Values

S_OK
: The object has changed since it was last saved.

S_FALSE
: The object has not changed since the last save.

Remarks

This method checks whether an object has changed since it was last saved so you can avoid losing information in objects that have not yet been saved. The dirty flag for an object is conditionally cleared in the **IPersistMemory::Save** method.

Notes to Callers

You should treat any error return codes as an indication that the object has changed. In other words, unless this method explicitly returns S_FALSE, you must assume that the object needs to be saved.

See Also

IPersistMemory::Save

IPersistMemory::Load

Instructs the object to load its persistent data from the memory pointed to by *pvMem* where *cbSize* indicates the amount of memory at *pvMem*. The object must not read past the address *(BYTE*)((BYTE *)pvMem+cbSize)*.

```
HRESULT Load(
    void *pvMem,        // Pointer to the stream from which the object should be loaded
    ULONG cbSize        // Amount of memory from which the object can read its data
  );
```

Parameters

pvMem
[in] The address of the memory from which the object can read up to *cbSize* bytes of its data.

cbSize
[in] The amount of memory available at *pvMem* from which the object can read its data.

Return Values

S_OK
The object successfully loaded its data.

E_UNEXPECTED
This method was called after the object was already initialized with **IPersistMemory::Load**. Only one initialization is allowed per instance.

E_POINTER
The pointer in *pvMem* is NULL.

Remarks

Any object that implements **IPersistMemory** has some information to load persistently, therefore E_NOTIMPL is not a valid return code.

See Also

IPersistMemory::InitNew

IPersistMemory::Save

Instructs the object to save its persistent data to the memory pointed to by *pvMem* where *cbSize* indicates the amount of memory available at *pvMem*. The object must not write past the address *(BYTE*)((BYTE *)pvMem+cbSize)*. The *fClearDirty* flag determines whether the object is to clear its dirty state after the save is complete.

```
HRESULT Save(
    void *pvMem,        // Pointer to the stream where the object is to be saved
    BOOL fClearDirty,   // Specifies whether to clear the dirty flag
    ULONG cbSize        // Amount of memory to which the object can write its data
);
```

Parameters

pvMem
[in] The address of the memory in which the object should save up to *cbSize* bytes of its data.

fClearDirty
[in] A flag indicating whether the object should clear its dirty state on return from **Save** or leave that state as-is.

cbSize
[in] The amount of memory available at *pvMem* to which the object can write its data.

Return Values

S_OK
The object successfully initialized itself.

E_UNEXPECTED
This method was called before the object was initialized with **IPersistMemory::InitNew** or **IPersistMemory::Load**.

E_INVALIDARG
The number of bytes indicated by *cbSize* is too small to allow the object to save itself completely.

E_POINTER
The pointer in *pvMem* is NULL.

Remarks

Any object that implements **IPersistMemory** has some information to save persistently, therefore E_NOTIMPL is not a valid return code.

The caller should ideally allocate as many bytes as the object returns from **IPersistMemory::GetSizeMax**.

See Also

IPersistMemory::InitNew, IPersistMemory::Load

IPersistMoniker

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

Existing moniker implementations call **QueryInterface** on the client objectfor persistence interfaces such as **IPersistFile**, **IPersistStream**[**Init**], or **IPersistStorage** as part of their **IMoniker::BindToObject** implementation when they are instantiating and initializing the object. The **IPersistMoniker** interface allows moniker implementations and other applications that instantiate objects from persistent data to give control to the object being instantiated over binding to its persistent data. An object could, for example, implement **IPersistMoniker::Load** by calling **IMoniker::BindToStorage** for the interface it prefers: **IStorage, IStream**, asynchronous binding, etc.

Unlike some other persistent object interfaces, **IPersistMoniker** does not include an **InitNew** method. This means that **IPersistMoniker** cannot be used to initialize an object to a freshly initialized state. Clients of **IPersistMoniker** who wish to initialize the object should **QueryInterface** for a different persistence interface that contains an **InitNew** method, such as **IPersistStreamInit**, **IPersistMemory**, or **IPersistPropertyBag**. Then, the client can use the **InitNew** method found in the other persistence interface to initialize the object. The client can still safely used **IPersistMoniker** to save the persistent state of the object.

The **IPersistMoniker** contract inherits its definition from the **IPersist** interface, and includes the **GetClassID** method of **IPersist**.

When to Implement

Implement **IPersistMoniker** on any object that can be saved persistently to multiple storage mediums or 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 most flexible persistence interface in their implementation of **IMoniker::BindToObject** if they are instantiating an 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**, and **IPersistMemory**.

Methods in Vtable Order

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

IPersist Method	Description
GetClassID	Returns the class identifier (CLSID) for the object.

IPersistMoniker Methods	Description
IsDirty	Checks an object for changes since it was last saved.
Load	Loads an object using a

	specified moniker.
Save	Saves the object, specifying a destination moniker.
SaveCompleted	Notifies the object that the save operation is complete.
GetCurMoniker	Gets the current moniker for the object.

IPersistMoniker::GetCurMoniker

Retrieves the moniker that refers to the object's persistent state.

HRESULT GetCurMoniker(
  **IMoniker *****ppmkCur* //Indirect pointer to moniker for the object's current persistent state.
);

Parameter

ppmkCur
  [out] Indirect pointer to the moniker that references the object's current persistent state.

Return Values

S_OK
  A valid absolute path was successfully returned.

E_INVALIDARG
  The *ppmkCur* parameter is invalid.

Remarks

Typically, this method returns the moniker last passed to the object via **IPersistMoniker::Load**, **IPersistMoniker::Save**, or **IPersistMoniker::SaveCompleted**.

See Also

IPersistMoniker::Load, IPersistMoniker::Save, IPersistMoniker::SaveCompleted

IPersistMoniker::IsDirty

Checks an object for changes since it was last saved.

HRESULT IsDirty(void);

Return Values

S_OK
: The object has changed since it was last saved.

S_FALSE
: The object has not changed since the last save.

Remarks

IPersistMoniker::IsDirty checks whether an object has changed since it was last saved so you can avoid losing information in objects that have not yet been saved.

See Also

IPersistMoniker::Save

IPersistMoniker::Load

Loads the object from its persistent state indicated by a supplied moniker.

```
HRESULT Load(
   BOOL fFullyAvailable,      //Indicates whether the object was already loaded
   IMoniker *pmkSrc,          //Pointer to source moniker that references the persistent state to be loaded
   IBindCtx *pbc,             //Pointer to the moniker's bind context
   DWORD grfMode              //Access mode values taken from the STGM enumeration
  );
```

Parameters

fFullyAvailable
[in] 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 FALSE, then an asynchronous bind operation should be launched.

pmkSrc
[in] Pointer to a moniker that references the persistent state for the object to be loaded.

pbc
[in] Pointer to the **IBindCtx** interface for the bind context to be used for any moniker binding during this method.

grfMode
[in] A combination of 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.

Return Values

S_OK
The object was successfully loaded.

E_INVALIDARG
One or more parameters are invalid.

Remarks

Typically, the object will immediately bind to its persistent state through a call to the source moniker's **IMoniker::BindToStorage** method, requesting either the **IStream** or **IStorage** interface.

See Also

IPersistMoniker::Save, IPersistMoniker::SaveCompleted

IPersistMoniker::Save

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

```
HRESULT Save(
    IMoniker *pmkDst,       //Pointer to destination moniker
    IBindCtx *pbc,          //Pointer to the moniker's bind context
    BOOL fRemember          //Specifies whether the destination moniker is to be the current working one
  );
```

Parameters

pmkDst

[in] Pointer to the moniker referencing the location where the object should persistently store itself. The object typically binds to the location through a call to *pmkDst->BindToStorage*, requesting either the **IStream** or **IStorage** interface. This parameter can be NULL, in which case the object should save itself to the same location referred to by the moniker passed to it in IPersistMoniker::Load. Using the NULL value, can act as an optimization to prevent the object from binding, since it has typically already bound to the moniker when it was loaded.

pbc

[in] Pointer to **IBindCtx** for the bind context to be used for any moniker binding during this method.

fRemember

[in] 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.

Return Values

S_OK

The object was successfully saved.

E_INVALIDARG

One or more parameters are invalid.

See Also

IPersistMoniker::GetCurMoniker, IPersistMoniker::Load, IPersistMoniker::SaveCompleted

IPersistMoniker::SaveCompleted

Notifies the object that it has been completely saved and points it to its new persisted state.

```
HRESULT SaveCompleted(
    IMoniker *pmkNew,      //Pointer to moniker for the object's new persistent state
    IBindCtx *pbc          //Bind context for binding during this method
  );
```

Parameter

pmkNew

[in] Pointer to the moniker for the object's new persistent state. This parameter can be NULL if the moniker to the object's new persistent state is the same as the previous moniker to the object's persistent state. This optimization is allowed only if there was a prior call to **IPersistMoniker::Save** with the *fRemember* parameter set to TRUE, in which case the object need not rebind to *pmkNew*.

pbc

[in] Pointer to the bind context to use for any moniker binding during this method.

Return Value

S_OK

The operation was successful.

E_INVALIDARG

One or more parameters are invalid.

Remarks

Typically, the object will immediately bind to its persistent state through a call to pmkNew->BindToStorage method, requesting either the **IStream** or **IStorage** interface, as in **IPersistMoniker::Load**.

See Also

IPersistMoniker::Load, IPersistMoniker::Save

IPersistPropertyBag

The **IPersistPropertyBag** interface works in conjunction with IPropertyBag and IErrorLog to define an individual property-based persistence mechanism. Whereas a mechanism like **IPersistStream** gives an object an **IStream** in which to store its binary data, **IPersistPropertyBag** provides an object with an **IPropertyBag** interface through which it can save and load individual properties. The implementor of **IPropertyBag** can then save those properties in whatever way it chooses, such as name/value pairs in a text file. Errors encountered in the process (on either side) are recorded in an error log through **IErrorLog**. This error reporting mechanism work on a per-property basis instead of an all properties as a whole basis through just the return value of **IPersist*::Load** or **IPersist*::Save**.

The basic mechanism is that a container tells the object to save or load its properties through **IPersistPropertyBag**. For each property, the object calls the container's **IPropertyBag** interface passed to the **IPersistPropertyBag** methods. **IPropertyBag::Write** saves a property in whatever place the container wants to put it, and **IPropertyBag::Read** retrieves a property.

This protocol is essentially a means of sequentially communicating individual property values from the object to the container, which is useful for doing save-as-text operations and the like. The object gives the container the choice of the format in which each property is saved, while retaining itself the decision as to which properties are saved or loaded.

Methods in Vtable Order

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

IPersist Method	Description
GetClassID	Returns the class identifier (CLSID) for the component object.

IPersistPropertyBag Methods	Description
InitNew	Informs the object that it is being initialized as a newly created object.
Load	Instructs the object to initialize itself using the properties in the property bag and notifying the error log when errors occur.
Save	Instructs object to save its properties to the specified property bag.

See Also

IErrorLog, IPropertyBag

IPersistPropertyBag::InitNew

Informs the object that it is being initialized as a newly created object.

HRESULT InitNew(void);

Return Values

S_OK
: The object successfully initialized itself. This should be returned even if the object doesn't do anything in the method.

E_UNEXPECTED
: This method was called after **IPersistPropertyBag::Load** or **IPersistPropertyBag::Save**.

Remarks

E_NOTIMPL should not be returned—use S_OK when the object has nothing to do in the method.

See Also

IPersistPropertyBag::Load

IPersistPropertyBag::Load

Instructs the object to initialize itself using the properties available in the property bag, notifying the provided error log object when errors occur. All property storage must take place within this method call as the object cannot hold the **IPropertyBag** pointer.

```
HRESULT Load(
    IPropertyBag *pPropBag,     // Points to caller's property bag
    IErrorLog *pErrorLog        // Points to error log
  );
```

Parameters

pPropBag

[in] A pointer to the caller's property bag through which the object can read properties. Cannot be NULL.

pErrorLog

[in] A pointer to the caller's error log in which the object stores any errors that occur during initialization. Can be NULL in which case the caller is not interested in errors.

Return Values

S_OK

The object successfully initialized itself.

E_UNEXPECTED

This method was called after **IPersistPropertyBag::InitNew** has already been called. They two initialization methods are mutually exclusive.

E_POINTER

The address in *pPropBag* is not valid (such as NULL) and therefore the object cannot initialize itself.

E_FAIL

The object was unable to retrieve a critical property that is necessary for the object's successful operation. The object was therefore unable to initialize itself completely.

Remarks

E_NOTIMPL is not a valid return code as any object implementing this interface must support the entire functionality of the interface.

See Also

IPersistPropertyBag::InitNew

IPersistPropertyBag::Save

Instructs the object to save its properties to the given property bag, optionally clearing the object's dirty flag. The caller can request that the object save all properties or that the object save only those that are known to have changed.

```
HRESULT Save(
    IPropertyBag *pPropBag,        // Points to the caller's property bag
    BOOL fClearDirty,              // Specifies whether to clear the dirty flag
    BOOL fSaveAllProperties        // Specifies whether to save all properties or just those that have changed
);
```

Parameters

pPropBag

[in] A pointer to the caller's property bag through which the object can write properties. Cannot be NULL.

fClearDirty

[in] A flag indicating whether the object should clear its dirty flag when saving is complete. TRUE means clear the flag, FALSE means leave the flag unaffected. FALSE is used when the caller wishes to do a Save Copy As type of operation.

fSaveAllProperties

[in] A flag indicating whether the object should save all its properties (TRUE) or only those that have changed since the last save or initialization (FALSE).

Return Values

S_OK

The object successfully saved the requested properties itself.

E_FAIL

There was a problem saving one of the properties. The object can choose to fail only if a necessary property could not be saved, meaning that the object can assume default property values if a given property is not seen through **IPersistPropertyBag::Load** at some later time.

E_POINTER

The address in *pPropBag* is not valid (such as NULL) and therefore the object cannot initialize itself.

Remarks

E_NOTIMPL is not a valid return code as any object implementing this interface must support the entire functionality of the interface.

See Also

IPersistPropertyBag::InitNew, IPersistPropertyBag::Load

IPersistStream - URL Moniker Implementation

The URL moniker implementation of **IPersistStream** is found on an URL moniker object, which supports **IUnknown**, **IAsyncMoniker**, and **IMoniker**. The **IMoniker** interface inherits its definition from **IPersistStream** and thus, the URL moniker also provides an implementation of **IPersistStream** as part of its implementation of **IMoniker**.

The **IAsyncMoniker** interface on an URL moniker is simply **IUnknown** (there are no additional methods); it is used to allow clients to determine if a moniker supports asynchronous binding. To get a pointer to the **IMoniker** interface on this object, call the CreateURLMoniker function. Then, to get a pointer to **IPersistStream**, call the **QueryInterface** method.

IPersistStream, in addition to inheriting its definition from **IUnknown**, also inherits the single method of **IPersist**, **GetClassID**.

When to Use

Call **IPersistStream** to have the moniker manage its persistent storage.

Remarks

IPersistStream::GetClassID
Returns CLSID_StdURLMoniker.

IPersistStream::IsDirty
Returns S_OK if the Moniker has changed since it was last saved (**IPersistStream::Save** with *fClearDirty*==TRUE), S_FALSE otherwise.

IPersistStream::Load
Initializes an URL moniker from data within a stream, usually stored there previously using its **IPersistStream::Save** (via **OleSaveToStream** or **OleSaveToStreamEx**). The binary format of the URL Moniker is its URL string in Unicode™ (may be a full or partial URL string, see CreateURLMoniker for details). This is represented as a ULONG (32-bit) count of characters followed by that many Unicode characters.

IPersistStream::Save
Saves an URL moniker to a stream. The binary format of URL Moniker is its URL string in Unicode (may be a full or partial URL string, see **CreateURLMoniker** for details). This is represented as a ULONG (32-bit) count of characters followed by that many Unicode™ characters.

IPersistStream::GetSizeMax
Returns the maximum number of bytes in the stream that will be required by a subsequent call to **IPersistStream::Save**. This value is SIZEOF(ULONG)==4 plus SIZEOF(WCHAR)**n* where *n* is the length of the full or partial URL string including the NULL terminator.

See Also

IMoniker - URL Moniker Implementation

IPrint

The **IPrint** interface enables OLE objects in general and document objects in particular to support programmatic printing. Once a document is loaded, containers and other clients can call IPrint::Print to instruct a document to print itself specifying printing control flags, the target device, the particular pages to print, and other options. The client can also control the continuation of printing by calling the IContinueCallback interface. In the future, additional print-related interfaces may be defined that will be available through **QueryInterface** where **IPrint** represents merely a base level of support.

An object that implements **IPrint** registers itself with the “Printable” key stored under its CLSID as follows:

```
HKEY_CLASSES_ROOT\CLSID\{...}\Printable
```

Callers determine whether a particular object class supports programmatic printing of its persistent state by looking in the registry for the “Printable” key.

When to Implement

Implementing **IPrint** is optional. You will usually implement **IPrint** on whatever object is able to load the persistent state of a given document type; that is, on the same object that also implements either **IPersistFile** or **IPersistStorage** for that type.

When to Use

Call **IPrint** when you want an object to print its current state. Using this interface, a caller can tell the object to print itself, set the initial page number of the printed document, and return both the number of pages and the number of the first page to be printed.

Methods in VTable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces
AddRef	Increments reference count
Release	Decrements reference count

IPrint Methods	**Description**
SetInitialPageNum	Sets number of first page
GetPageInfo	Returns number of pages and number of first page
Print	Prints object’s persistent state

IPrint::GetPageInfo

Retrieves the number of a document’s first page as well as the total number of pages.

HRESULT GetPageInfo(
  LONG ****nFirstPage*,** // Pointer to number of first page
  LONG ****pcPages // Pointer to total number of pages
);

Parameters

pnFirstPage
  [out] Pointer to the page number of the first page. May be NULL, indicating the caller doesn’t need this number. If IPrint::SetInitialPageNum has been called, this parameter should contain the same value passed to that method. Otherwise, the value is the document's internal first page number.

pcPages
  Pointer to the total number of pages in this document. May be NULL, indicating the caller doesn’t need this number.

Return Value

S_OK
  The first page was set as requested.

E_UNEXPECTED
  An unknown error occurred.

IPrint::Print

Prints an object on the specified printer, using the specified job requirements.

```
HRESULT Print(
    DWORD grfFlags,                 // A bit field
    DVTARGETDEVICE **pptd,          // On return, Indirect pointer to the target device
    PAGESET **pppageset,            // On return, Indirect pointer to the set of pages to be printed
    STGMEDIUM *pstgmOptions,        // Pointer to object-specific printing options
    IContinueCallback *pcallback,   // Pointer to callback interface
    LONG nFirstPage,                // Number of first page to be printed
    LONG *pcPagesPrinted,           // Pointer to total number of pages printed
    LONG *pnLastPage                // Pointer to number of last page to be printed
  );
```

Parameters

grfFlags
[in] A bit field whose values are taken from the enumeration PRINTFLAG.

pptd
[in, out]The target device on which the printing is to occur.

ppPageSet
[in, out] Indicates which pages are to be printed.

ppstgmOptions
[unique][in, out] Contains object-specific printing options in the form of a serialized OLE property set. May be NULL in one or both directions.

pCallback
[in] A callback interface which is to be periodically polled at human-response speeds to determine whether printing should be abandoned. May be NULL.

nFirstPage
[in] The page number of the first page to be printed. This value overrides any value previously passed to IPrint::SetInitialPageNum.

pcPagesPrinted
[out] The place at which the object is to return the actual number of pages that were successfully printed.

pnLastPage
[out] The place at which the object is to return the last legal page number.

Return Values

S_OK
Success

PRINT_E_CANCELED
The print process was canceled. **pcPagesPrinted* indicates the number of pages that were in fact successfully printed before this error occurred.

PRINT_E_NOSUCHPAGE
An attempt has been made to print a page which does not exist.

E_UNEXPECTED
An unexpected error occurred.

Remarks

The printer on which the object is to be printed is indicated by the **DVTARGETDEVICE** structure in *ptd*. The **DEVMODE** structure in the target device indicates whole-job printer-specific options, such as number of copies, paper size, print quality, and other options. The DEVMODE structure may or may not also contain orientation information in the *dmOrientation* field (this is indicated in the *dmFields* field). If present, then this paper orientation should be used; if absent, then natural orientation as determined by the object content is to be used.

Due to the possibility of user input, the parameters *pptd* and *ppPageSet* are both [in,out] structures. In the absence of user interaction (that is, without PRINTFLAG_PROMPTUSER), both the target device and the page set will necessarily be the same for input and output. However, if the user is prompted for print options, then the object returns target device and page-set information appropriate to what the user has actually chosen.

The *ppstgmOptions* parameter is also [in,out]. On exit, the object should return through **ppstgmOptions* any object-specific information that it would need to reproduce this exact print job. Examples might include whether the user selected “sheet, notes, or both” in a spreadsheet application. The data returned is in the format of a serialized property set. The returned data is normally useful only when passed back in a subsequent call to the same object. Because a subsequent call may specify different user interaction flags, target device, or other settings, the caller can cause the exact same document to be printed multiple times in slightly different printing contexts.

See Also

TBD

IPrint::SetInitialPageNum

Attempts to set the number of the first page of a document.

```
HRESULT SetInitialPageNum(
    LONG nFirstPage      // Number of first page
  );
```

Parameters

nFirstPage
[in] The desired first page number

Return Value

S_OK
The first page was set as requested.

E_FAIL
The first page could not be set to the indicated value.

E_UNEXPECTED
An unknown error occurred.

Remarks

Note that setting a negative first page number is legal: this may be useful in printing a portion of the document with offset page numbers from what it would normally print. Note also that not all implementations permit the initial page number to be set, as some implementations simply lack the information as to how this page information should be reflected in the final output.

IPropertyBag

When a client wishes to have exact control over how individually named properties of an object are saved, it would attempt to use an object's **IPersistPropertyBag** interface as a persistence mechanism. In that case the client supplies a property bag to the object in the form of an **IPropertyBag** interface.

When the object wishes to read a property in **IPersistPropertyBag::Load** it will call **IPropertyBag::Read**. When the object is saving properties in **IPersistPropertyBag::Save** it will call **IPropertyBag::Write**. Each property is described with a name in *pszPropName* whose value is exchanged in a **VARIANT**. This information allows a client to save the property values as text, for instance, which is the primary reason why a client might choose to support **IPersistPropertyBag**.

The client records errors that occur during reading into the supplied error log.

Methods in Vtable Order

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

IPropertyBag Methods	Description
Read	Asks the property bag to read the specified property into a caller-initialized **VARIANT**.
Write	Asks the property bag to save the specified property using the type and value in the caller-initialized **VARIANT**.

See Also

IErrorLog, IPersistPropertyBag

IPropertyBag::Read

Asks the property bag to read the property named with *pszPropName* into the caller-initialized **VARIANT** in *pVar*. Errors that occur are logged in the error log pointed to by *pErrorLog.* When **pVar->vt** specifies another object pointer (VT_UNKNOWN) then the property bag is responsible for creating and initializing the object described by *pszPropName*.

```
HRESULT Read(
   LPCOLESTR   pszPropName,     // Points to the property to be read
   VARIANT   * pVar,            // Points to the VARIANT to receive the property value
   IErrorLog * pErrorLog        // Points to the caller's error log
  );
```

Parameters

pszPropName

[in] The name of the property to read. Cannot be NULL.

pVar

[in, out] The address of the caller-initialized **VARIANT** that is to receive the property value on output. The method must set both type and value fields in the **VARIANT** before returning. If the caller initialized the **pVar->vt** field on entry, the property bag should attempt to coerce the value it knows into this type. If the caller sets **pVar->vt** to VT_EMPTY, the property bag can use whatever type is convenient.

pErrorLog

[in] A pointer to the caller's error log in which the property bag stores any errors that occur during reads. Can be NULL in which case the caller is not interested in errors.

Return Values

S_OK

The property was read successfully. The caller becomes responsible for any allocations that are contained in the **VARIANT** in *pVar*.

E_POINTER

The address in *pszPropName* is not valid (such as NULL).

E_INVALIDARG

The property named with *pszPropName* does not exist in the property bag.

E_FAIL

The property bag was unable to read the specified property, such as if the caller specified a data type to which the property bag could not coerce the known value. If the caller supplied an error log, a more descriptive error was sent there.

Remarks

E_NOTIMPL is not a valid return code since any object implementing this interface must support the entire functionality of the interface.

See Also

IPropertyBag::Write

IPropertyBag::Write

Asks the property bag to save the property named with *pszPropName* using the type and value in the caller-initialized **VARIANT** in *pVar.* In some cases the caller may be asking the property bag to save another object, that is, when **pVar->vt** is VT_UNKNOWN. In such cases, the property bag queries this object pointer for some persistence interface, like **IPersistStream** or even **IPersistPropertyBag** again and has that object save its data as well. Usually, this results in the property bag having some byte array for this object which can be saved as encoded text (hex string, MIME, etc.). When the property bag is later used to reinitialize a control, the client that owns the property bag must recreate the object when the caller asks for it, initializing that object with the previously saved bits.

This allows very efficient persistence operations for large BLOB properties like a picture, where the owner of the property bag itself directly asks the picture object (managed as a property in the control being saved) to save into a specific location. This avoids potential extra copy operations that would be involved with other property-based persistence mechanisms.

```
HRESULT Write(
  LPCOLESTR   pszPropName,    // Points to the property to be written
  VARIANT   * pVar            // Points to the VARIANT containing the property value and type
);
```

Parameters

pszPropName

[in] The name of the property to write. Cannot be NULL.

pVar

[in] The address of the caller-initialized **VARIANT** that holds the property value to save. The caller owns this **VARIANT** and is responsible for all allocations therein. That is, the property bag itself does not attempt to free data in the **VARIANT**.

Return Values

S_OK

The property bag successfully saved the requested property.

E_FAIL

There was a problem writing the property. It is possible that the property bag does not understand how to save a particular **VARIANT** type.

E_POINTER

The address in *pszPropName* or *pVar* is not valid (such as NULL). The caller must supply both.

Remarks

E_NOTIMPL is not a valid return code as any object implementing this interface must support the entire functionality of the interface.

See Also

IPropertyBag::Read

IProvideClassInfo2

IProvideClassInfo2 is a simples extension to **IProvideClassInfo** for the purpose of making it quick and easy to retrieve an object's outgoing interface IID for its default event set. The mechanism, the added **GetGUID** method, is extensible for other types of GUIDs as well.

Methods in Vtable Order

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

IProvideClassInfo Method	Description
GetClassInfo	Returns the **ITypeInfo** interface for the object's **coclass** type information.

IProvideClassInfo2 Method	Description
GetGUID	Returns the GUID for the object's outgoing IID for its default event set.

IProvideClassInfo2::GetGUID

Returns a GUID corresponding to the specified *dwGuidKind*. The *dwGuidKind* parameter has several values defined. See GUIDKIND. Additional flags can be defined at a later time and will be recognized by an **IProvideClassInfo2** implementation. See IProvideClassInfo3.

```
HRESULT GetGUID(
    DWORD dwGuidKind,      // Specifies the desired GUID
    GUID *pGUID            // Receives a pointer to the desired GUID
  );
```

Parameters

dwGuidKind
[in] Specifies the GUID desired on return. This parameter takes a value from the **GUIDKIND** enumeration.

pGUID
[out] The address of the caller's variable in which to store the GUID associated with *dwGuidKind*.

Return Values

S_OK
The GUID was successfully returned in **pGUID*.

E_POINTER
The address in *pGUID* is not valid (such as NULL).

E_UNEXPECTED
An unknown error occurred.

E_INVALIDARG
The *dwGuidKind* value does not correspond to a supported GUID kind.

Remarks

E_NOTIMPL is not a valid return code since it would be pointless to implement this interface without implementing this method. E_INVALIDARG is not valid when *dwGuidKind* is GUIDKIND_DEFAULT_SOURCE_DISP_IID and the object is not implementing **IProvideClassInfo2** as part of its implementation of **IProvideClassInfo3**.

Any object implementing **IProvideClassInfo3** should support at least GUIDKIND_DEFAULT_IID through this method, so E_NOTIMPL is not a valid return code for **IProvideClassInfo3** as it is for **IProvideClassInfo2**.

See Also

GUIDKIND, IProvideClassInfo3

IProvideClassInfo3

This interface is an extension to **IProvideClassInfo2** to provide various pieces of object-level information directly from the object (as opposed to using the registry or some other means that does not involve the object). As such it is intended to be a generic object-information interface. This interface also corrects the absence of an LCID in **IProvideClassInfo::GetClassInfo**.

Methods in Vtable Order

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

IProvideClassInfo Method	Description
GetClassInfo	Returns the **ITypeInfo** interface for the object's **coclass** type information.

IProvideClassInfo2 Method	Description
GetGUID	Returns the GUID for the object's outgoing IID for its default event set.

IProvideClassInfo3 Method	Description
GetGUIDDwordArrays	Provides a caller with easy access to a substantial amount of object information that generally exists or can be expressed at arrays of GUIDs and/or DWORDs.
GetClassInfoLocale	Returns an LCID locale identifier for the specified type.
GetFlags	Returns a DWORD containing up to 32 mutually-inclusive (generally) bit flags that are all related to the flag group identified by *guidGroup*.

See Also

GUIDKIND, IProvideClassInfo2::GetGUID

IProvideClassInfo3::GetClassInfoLocale

Returns a pointer to the **ITypeInfo** interface for the object's type information. The type information for an object corresponds to the object's **coclass** entry in a type library. This method also uses a locale identifier (LCID) to specify the locale so that locale-specific type information is returned. This method is similar to **IProvideClassInfo::GetClassInfo**.

HRESULT GetClassInfoLocale(
  ITypeInfo *****ppTypeInfo*,** // Receives an indirect pointer to the **ITypeInfo** interface
  LCID *lcid* // Specifies the locale identifier
);

Parameters

ppTypeInfo
[out] The address of the caller's **ITypeInfo** * variable in which to store the object's type information. The caller is responsible for calling **ITypeInfo::Release** on the returned pointer if this method returns successfully.

lcid
[in] Specifies the locale.

Return Values

S_OK
The type information was successfully returned.

E_POINTER
The address in *ppTypeInfo* is not valid. For example, it may be NULL.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The type information could not be retrieved.

Remarks

The caller is responsible for calling **ITypeInfo::Release** when the returned interface pointer is no longer needed.

IProvideClassInfo3::GetFlags

Returns a DWORD containing up to 32 mutually-inclusive (generally) bit flags that are all related to the flag group identified by *guidGroup*. This method is extensible as the owner of a particular group can add new bit flags at any time to that group (provided there are enough bits remaining) and that anyone can create a new group GUID and assign flags as desired.

This method is designed to handle many future uses of bit flags that have traditionally been limited to the MiscStatus bits as returned from **IOleObject::GetMiscStatus**. At the time of writing, only one group has been defined, FLAGID_Internet, with only one bit in that group, INTERNETFLAG_USESDATAPATH. This bit indicates that the object uses one or more data path properties to manage BLOB data storage. See INTERNETFLAG.

HRESULT GetFlags(
  REFGUID *guidGroup***,** // Specifies the flag group requested by the caller
  DWORD **pdwFlags* // Receives a pointer to the flag values
);

Parameters

guidGroup
[in] Unique identifier of the flag group desired by the caller.

pdwFlags
[out] The address of the caller's variable in which to store the DWORD containing the bit flags associated with *guidGroup*.

Return Values

S_OK
The DWORD was successfully returned in *pdwFlags.

E_POINTER
The address in *pdwFlags* is not valid (such as NULL).

E_UNEXPECTED
An unknown error occurred.

E_INVALIDARG
The *guidGroup* value does not correspond to a group known by this object.

E_NOTIMPL
The object supports no flag groups and therefore has no reason to implement this method.

IProvideClassInfo3::GetGUIDDwordArrays

Provides a caller with easy access to a substantial amount of object information that generally exists or can be expressed at arrays of GUIDs and/or DWORDs. Through this method a caller can obtain an array of GUIDs, an array of DWORDs, or both. When this method returns information in both arrays, then there is a relationship between the elements of both arrays, that is, *pcaUUID[0]* and *pcadw[0]* are related in some way, depending on the kind of array in question as described by *rguidArray*.

HRESULT GetGUIDDwordArrays(
REFGUID *rguidArray***,** // Specifies requested GUID or DWORDs
CAUUID ****pcaUUID*,** // Points to CAUUID structure in which to store requested GUIDs
**CADWORD ****pcadw* // Points to CADWORD structure in which to store requested DWORDs
);

Return Values

rguidArray

[in] Identifier of the GUIDs or DWORDs desired on return. This parameter can have the following values. Additional array types can be defined by anyone at any time by simply defining another ARRAYID for use with this method.

ARRAYID_Interfaces_Incoming

On return, *pcaUUID* contains the interface identifiers of all the object's possible incoming interfaces that a client might access through **QueryInterface**. The *pcadw* parameter in this case must be empty (that is, **pcadw.cElems** is zero). These IIDs are only suggestions of what the object supports and in no way guarantee support at run-time. A client must always request an interface through **QueryInterface(riid)** and must be able to handle the E_NOINTERFACE failure code even if *riid* was specified in this array.

ARRAYID_Interfaces_Outgoing

On return, *pcaUUID* contains the interface identifiers of all the object's outgoing interfaces, that is, those that would be marked as **[source]** in an object's type library. The *pcadw* parameter in this case must be empty (that is, **pcadw.cElems** is zero). These IIDs are only suggestions of what the object supports and in no way guarantee support at run-time. A client must always connect to an outgoing interface through **IConnectionPointContainer::FindConnectionPoint(riid)** must handle the possible failure of this call even if *riid* in question was specified in this array.

ARRAYID_Categories_Implemented

On return, *pcaUUID* contains the CATIDs of the object's supported categories. The *pcadw* parameter in this case must be empty (that is, **pcadw.cElems** is zero).

ARRAYID_Categories_Required

On return, *pcaUUID* contains the CATIDs of the object's required categories, that is, the necessary container-side support without which the object cannot operate. The *pcadw* parameter in this case must be empty (that is, **pcadw.cElems** is zero).

ARRAYID_PathProperties

On return, *pcadw* contains the dispIDs of the object's data path properties, and *pcaUUID* contains the GUIDs identifying the **OLE_DATAPATH_<xx>** type of the corresponding property in *pcadw*. The object must allocate and fill both arrays in this case, and both arrays must container the same number of elements.

ARRAYID_Methods_Primary

On return, *pcadw* contains the dispIDs of the object's primary methods that best describe its user-level functionality—such methods would be displayed in the caller's user interface for providing features like event binding. In this case, **pcaUUID.cElems** must be zero.

ARRAYID_Methods_Secondary

On return, *pcadw* contains the dispIDs of the object's seconday methods that is exactly the list of

all object methods excluding those returned from ARRAYID_Methods_Primary. In this case, **pcaUUID.cElems** must be zero.

pcaUUID
[in, out] The address of the caller-allocated and initialized **CAUUID** structure in which to store the GUIDs associated with *rguidArray*, if applicable. The caller is responsible for freeing any allocated memory returned in this structure.

pcadw
[in, out] The address of the caller-allocated and initialized **CADWORD** structure in which to store the DWORDs associated with *rguidArray*, if applicable. The caller is responsible for freeing any allocated memory returned in this structure.

Return Values

This method can return several success codes that identify which array was or was not filled. This provides an alternate means for the caller to identify which arrays contain allocated data and which do not. On failure the object must return zero and NULL in the *cElems* and *pElems* fields of both structures, respectively. That is, the caller need not free any allocations on failure.

S_OK
Both **CAUUID** and **CADWORD** arrays were successfully allocated and filled.

CLASSINFO_S_ONLYGUIDS
Only the **CAUUID** array was allocated and filled whereas the **CADWORD** array was not needed. In this case **CADWORD::cElems** must be zero and **CADWORD::pElems** must be NULL.

CLASSINFO_S_ONLYDWORDS
Only the **CADWORD** array was allocated and filled whereas the **CAUUID** array was not needed. In this case **CAUUID::cElems** must be zero and **CAUUID::pElems** must be NULL.

E_OUTOFMEMORY
One of the arrays could not be allocated.

E_POINTER
The address in either *pcaUUID* or *pcadw* is not valid (such as NULL).

E_UNEXPECTED
An unknown error occurred.

E_INVALIDARG
The *rguidArray* value does not correspond to a supported array.

Remarks

E_NOTIMPL is not a valid return code for this method as all objects can at least support ARRAYID_Interfaces_Incoming.

The data structures **CAUUID** and **CADWORD** were originally defined for other OLE Controls interfaces like **ISpecifyPropertyPages** and **IPerPropertyBrowsing**. These structures are defined as follows:

```
typedef struct tagCAUUID
    {
    ULONG  cElems;
    GUID  *pElems;
    } CAUUID;

typedef struct tagCADWORD
    {
    ULONG  cElems;
    DWORD *pElems;
    } CADWORD;
```

In these definitions, *cElems* is the number of entries in the array pointed to by *pElems*. The structures themselves are caller-allocated and caller-initialized so *cElems* is zero and *pElems* is NULL. The structures are filled on output by the object implementing the method, where *cElems* describes the number of elements that the object stored in *pElems*, where the *pElems* array itself is object-allocated using *CoTaskMemAlloc*. On successful return, the caller is responsible for freeing any allocated memory in these structures. That is, if *pElems* is non-NULL (*cElems* is non-zero) then the caller must call *CoTaskMemFree(pElems)* when the array is no longer needed.

IServiceProvider

IServiceProvider locates a service specified by its GUID and returns the interface pointer for the requested interface on the service.

When to Implement

An object that provides services should implement the **IServiceProvider** interface as a general way to supply its clients with interface pointers to the interfaces on the service.

A service is often provided through a separate object from the client site. For example, the service can be provided through a separate control or some other object that the client can communicate with.

Usually, the client communicates through its client site object in the container. The container calls **IOleObject::SetClientSite** to provide a pointer to the **IOleClientSite** interface for the embedded object's client site. Then, the client must call methods in the **IOleClientSite** interface to find out about services that its container supports. Thus, the client site must provide a way for the client to access the service when necessary, even if the service is provided through a separate object.

For example, an in-place object calls **IOleInPlaceSite::GetWindowContext** to obtain interface pointers for the document object that contains the site and for the frame object that contains the document. Both of these interface pointers are on objects separate from the site object, so the client cannot call **IOleInPlaceSite::QueryInterface** to obtain these interface pointers.

The **IServiceProvider** interface is a general way to provide interface pointers for services, so that the site object need not implement ad hoc solutions as the need arises.

When to Use

This interface itself has only one method, **IServiceProvider::QueryService**. The caller specifies a GUID for the service and the IID of the requested interface. The interface pointer is returned in a caller-supplied variable.

Methods in Vtable Order

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

IServiceProvider Methods	Description
QueryService	Returns an interface pointer to the requested interface on a service.

See Also

IServiceProvider::QueryService

Creates or accesses the specified service and returns an interface pointer to the specified interface for the service.

```
HRESULT QueryService(
   REFGUID guidService,     //Unique identifier for the requested service
   REFIID riid,             //Unique identifier for the requested interface
   void **ppv               //Indirect pointer to the interface on the service
  );
```

Parameter

guidService
   [in] Unique identifier for the requestedservice.

riid
   [in] Unique identifier for the requested interface on the service.

ppv
   [out] Indirect pointer to the interface on the service.

Return Values

S_OK
   The service was successfully created or retrieved.

E_OUTOFMEMORY
   There is not enough memory to create the service.

E_UNEXPECTED
   An unknown error occurred.

E_NOINTERFACE
   The service exists but the requested interface is not provided by the service.

SVC_E_UNKNOWNSERVICE
   The service identified with *guidService* is not recognized.

Note to Callers

The caller is responsible for releasing the interface pointer when it is no longer needed.

Note to Implementors

Because there is only one method in this interface, E_NOTIMPL is not a valid return code.

IWindow

During a bind operation, the moniker can obtain some additional bind information by calling the bind client's **IBindStatusCallback::GetBindInfo**. In some cases, additional callback interfaces can be provided by the client requesting the bind operation to supply even more services to the moniker performing the bind. These additional interfaces are typically requested by the moniker calling **IBindStatusCallback::QueryInterface** for the interface desired. However, a moniker client can also provide additional interfaces through **IServiceProvider**.

In the case of URL monikers, the **IWindow** interface is an additional callback interface that the client can provide.

When to Implement

Clients of URL monikers implement this interface to allow the URL moniker to display information in the client's user interface when necessary. This interface is usually implemented by OLE containers that also implement the **IBindHost** interface.

When to Use

The URL moniker uses **IWindow** to display information in the client's user interface. Note that the URL moniker will not request **IWindow** from clients unless the bind context BIND_OPTS specify the *grfFlags* value of BIND_MAYBOTHERUSER. The bind context BIND_OPTS is set by calling **IBindCtx::SetBindOptions**.

Methods in Vtable Order

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

IWindow Methods	Description
GetWindow	Returns an HWND for displaying user interface information.

IWindow::GetWindow

The URL moniker calls this method to obtain an **HWND** for a window to present information in the user interface during a bind operation.

HRESULT GetWindow(
  HWND ***phwnd** //Pointer to window handle of parent window for moniker to use
);

Parameter

phwnd
  [out] Pointer to a window handle of the parent window to use for displaying user interface information.

Return Values

S_OK
  The window handle was successfully returned.

E_INVALIDARG
  The *phwnd* parameter is invalid.

Notes to Implementors

If you implement this interface, you can return S_FALSE for this method to indicate that no window is available for displaying user interface information.

See Also

IAuthenticate, IHttpNegotiate, IWinINetInfo

IWinINetInfo

The **IWinINetInfo** interface provides protocol-specific information for a bind operation.

When to Implement

The binding object that exposes the **IBinding** interface implements **IWinINetInfo** for standard Internet protocols.

When to Use

The client of an URL moniker calls this interface obtaining a pointer from **IBinding::QueryInterface**. Then, the moniker client can query for protocol-specific information from the binding object.

Methods in Vtable Order

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

IWinINetInfo Methods	Description
QueryOption	Queries for protocol-specific information about the bind operation.

IWinINetInfo::QueryOption

The moniker client calls this method to query for protocol-specific information about the bind operation.

```
HRESULT QueryOption(
    DWORD   dwOption,     //Information requested
    LPVOID  pBuffer,      //Buffer to receive query results
    DWORD   *pcbBuf       //Pointer to size of buffer
  );
```

Parameters

dwOption

[in] Information being requested. Valid values are documented with **InternetQueryOption**.

pBuffer

[out] Pointer to the buffer to receive the results of the query.

pcbBuf

[in, out] Pointer to the **DWORD** value for the length of data written to *pBuffer*.

Return Values

S_OK

The query was successful.

E_INVALIDARG

One or more parameters are invalid.

Remarks

This method maps directly to the Windows Internet **InternetQueryOption** function.

CoGetClassObjectFromURL

Returns a factory object for a given *rclsid*. If no **CLSID** is specified (**CLSID_NULL**), this function will choose the appropriate **CLSID** for interpreting the Internet mail extensions (MIME) type specified in *szContentType*. If the desired object is installed on the system, it is instantiated. Otherwise, the necessary code is downloaded and installed from the location specified in *szCodeURL*.

```
STDAPI CoGetClassObjectFromURL(
    REFCLSID rclsid,          //CLSID of the ActiveX object to be installed
    LPCWSTR szCodeURL,        //URL pointing to the code for the ActiveX object
    DWORD dwFileVersionMS,    //Major version number for object to be installed
    DWORD dwFileVersionLS,    //Minor version number for object to be installed
    LPCWSTR szContentType,    //Mime type to be understood by the installed ActiveX object
    LPBINDCTX pBindCtx,       //Bind context to use for downloading/installing component code
    DWORD dwClsContext,       //Specifies the execution context for the class object
    LPVOID pvReserved,        //Reserved, must be set to NULL
    REFIID riid,              //Interface to obtain on the factory object
    VOID ppv * *              //For synchronous calls, pointer to store the interface pointer upon return
  );
```

Parameters

rclsid

[in] The **CLSID** of the ActiveX object to be installed. If the value is CLSID_NULL, *szContentType* is used to determine the **CLSID**.

szCodeURL

[in] The URL pointing to the code for the ActiveX object. This can point to a “portable executable” (.OCX, .DLL, .EXE), to an .CAB (cabinet) file, or to a .INF file. If this value is NULL, the Internet Component Download will still attempt to download the desired code from an Object Store on the Internet Search Path.

dwFileVersionMS

[in] The major version number for the object to be installed. If this value and the value for *dwFileVersionLS* are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired. This means that Internet Component Download will always attempt to download new code.

dwFileVersionLS

[in] The minor version number for the object to be installed. If this value and the value for dwFileVersionMS are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired. This means that Internet Component Download will always attempt to download new code.

szContentType

[in] The MIME type to be understood by the installed ActiveX object. If *rclsid* is CLSID_NULL, this string is used to determine the **CLSID** of the object to be installed.

pBindCtx

[in] The bind context to use for downloading/installing component code. Register **IBindStatusCallback** in this bind context to receive callbacks during the download and installation process.

dwClsContext

[in] Values taken from the **CLSCTX** enumeration specifying the execution context for the class object.

pvReserved
[in] Reserved, must be set to NULL.

riid
[in] The interface to obtain on the factory object. Usually, this interface is **IClassFactory**.

ppv
[out] Upon return for synchronous calls, the pointer in which to store the interface pointer.

Return Values

S_OK
The function was successful; the *ppv* contains the requested interface pointer.

MK_S_ASYNCHRONOUS
Component code will be downloaded and installed asynchronously. The client will receive notifications through the **IBindStatusCallback** interface registered on *pBindCtx*.

E_NOINTERFACE
The desired interface pointer is not available.

Remarks

Because this function was designed to enable component download in Web browsers, the parameters passed to **CoGetClassObjectFromURL** closely match the values expressed in the HTML OBJECT tag. For example, *szCodeURL*, *dwFileVersionMS*, and *dwFileVersionLS* are specified inside an OBJECT tag as:

```
code=http:-/-/www.foo.com/bar.ocx#Version=a,b,c,d
```

where:

szCodeURL is HTTP://WWW.FOO.COM/BAR.OCX

cdFileVersionMS is MAKELONG(B,A)

dwFileVersionLS is MAKELONG(D,C)

If the *pBindCtx* parameter to the function is an asynchronous bind context created by **CreateAsyncBindContext**, downloading and installation of code occurs asynchronously, and **CoGetClassObjectFromURL** will return immediately with a return value of MK_S_ASYNCHRONOUS. Then, the IBindStatusCallback registered on the asynchronous bind context is used to communicate the status of the download operation to the client. By implementing the ICodeInstall callback, the client can participate in further negotiation for the download process. To participate in this communication, implement **IBindStatusCallback** and register this interface in *pBindCtx* passed to **CoGetClassObjectFromURL** using RegisterBindStatusCallback. Callback notifications will occur from the following:

IBindStatusCallback::OnStartBinding
Provides an **IBinding** for controlling the download process.

IBindStatusCallback::OnProgress
Reports progress on the download process.

IBindStatusCallback::QueryInterface
Requests additional interfaces (for example, **ICodeInstall**) for further negotiation.

IBindStatusCallback::OnObjectAvailable
Returns the desired object interface pointer.

IBindStatusCallback::OnStopBinding
Returns any error codes.

Note The initial implementation of **CoGetClassObjectFromURL** does not support system-wide simultaneous downloading of the same code. Similarly, it will not support cases by which different but simultaneous downloading refers to the same piece of dependent code.

Packaging Component Code for Automatic Download

There are three ways to package code for download:

- As a single PE (portable executable) format file
- By using a .CAB file
- By using a stand-alone .INF file

A single PE is the simplest way to package a single-file OLE control. This single file is downloaded, installed, and registered in one operation. Note, however, that this method will not use file compression and it will not be platform independent, except with HTTP. No packaging is done with this method, the control is simply placed on a server.

Another alternative to the PE method is to package the control in a .CAB file. Cabinet files are archives of one or more files compressed using Lempel-Ziv compression. Using a .CAB file offers two advantages over the PE format method. .CAB files are compressed, which reduces the time required to download the control. In addition, multiple files can be downloaded. To package the control in a .CAB file, collect all the required files and write an .INF file to provide further installation instructions. The .INF file refers to files in the .CAB and to files at other URLs.

The third method of packaging is to specify the .INF file that contains directions for downloading and installing the control, rather than specifying the .OCX file directly or by specifying a .CAB file. The primary advantage of this method is that it provides platform independence. The .INF file is downloaded first. It is possible to specify files to download for different target platforms in the .INF file. When the .INF file is interpreted, the appropriate set of files can be downloaded. Using .INF files also offers the most flexibility in downloading the minimum amount of code required to get the control to work. Note, however, that this method does not provide the compression offered by the .CAB method. The following example is used to create a platform-independent .INF file:

```
[circ3.ocx]
file_win32_x86=file://products/release/circ3/x86/circ3.cab
file_win32_mips=file://products/release/circ3/mips/circ3.cab
file_mac_ppc=ignore
```

In the example above, the specified component circ3.ocx (CLSID, Version) needs to be installed on the system. If not already in existence, it can be downloaded from the given location (a .CAB). The **ignore** keyword indicates that this file is not needed for this platform.

Internet Search Path

When Internet Component Download is called to download code, it traverses the Internet Search Path to look for the desired component. This path is a list of Object Store Servers that will be queried every time components are downloaded using **CoGetClassObjectFromURL**. In this way, even if an OBJECT tag in an HTML document does not specify a **CODEBASE** location to download code for an embedded OLE control, the Internet Component Download will still use the Internet Search Path to find the required code. The Internet Search Path provides a great deal of flexibility in determining how files get downloaded. By setting up URLs in the path before the **CODEBASE** keyword, files can be installed from local intranet caches of common components. By removing the **CODEBASE** keyword from the path completely, component download from the internet is effectively disabled. By setting up URLs in the path after the **CODEBASE** keyword, files can be installed from standard distribution points, even if the server specified is not available.

The version number is also considered when locating files. If a version number is specified, the

download service will only try to download and install the file if the version number specified is more recent that any version currently installed on the system. If the version number specified is **-1**, **-1**,**-1**,**-1**, the download service will always try to locate the latest version of the file for download. When searching the Internet Search Path, each Object Store receives an HTTP POST request containing the CLSID or MIME type and, optionally, a version number in the following format:

```
CLSID={class id}
Version=a,b,c,d
MIMETYPE=mimetype
```

All of the values are optional, however, either CLSID or MIMETYPE must be present. The Object Store parses this information, checks the internal database of available files, and redirects the HTTP request to the appropriate downloadable file.

The search path is specified in a string in the Registry, under the following key:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\InternetSettings\CodeBaseSearchPath
```

The value for this key is a string in the following format:

```
CodeBaseSearchPath=<URL (1) >;<URL (2) >;...<URL (m) >;CODEBASE;<URL (m+1) >;...<URL (n-1) >;<URL (n) >
```

In the example above, each URL is an absolute URL pointing to HTTP servers acting as Object Stores. When processing a call to **CoGetClassObjectFromURL**, the Internet Component Download service will first try downloading the desired code from the locations URL $_{(1)}$ through URL $_{(m)}$, it will then try the location specified in the *szCodeURL* parameter, and will finally try the locations specified in locations URL $_{(m+1)}$ through URL $_{(n)}$.

Note If the **CODEBASE** keyword is not included in the CodeBaseSearchPath key, calls to **CoGetClassObjectFromURL** will not check the *szCodeURL* location for downloading code.

See Also

IBindStatusCallback, IBindStatusCallback::OnStartBinding, IBindStatusCallback::OnProgress, IBindStatusCallback::OnObjectAvailable, IBindStatusCallback::OnStopBinding, ICodeInstall, RegisterBindStatusCallback

CreateAsyncBindCtx

Creates an asynchronous bind context for use with asynchronous monikers.

```
HRESULT CreateAsyncBindCtx(
  DWORD   dwReserved,              //Reserved for future use; must be zero
  IBindStatusCallback *pbsc,       //Pointer to callback interface
  DWORD   grfBSCOption,            //BSCO_OPTION flags
  IEnumFORMATETC*penumfmtetc,      //Pointer to enumerator object for formats
  IBindCtx **ppbc                  //Indirect pointer to new bind context
 );
```

Parameters

dwReserved
[in] Reserved for future use; must be zero.

pbsc
[in] Pointer to the **IBindStatusCallback** interface used for receiving data availability and progress notification.

grfBSCOption
[in] Flags from the **BSCO_OPTION** enumeration specifying which callback notifications the client wants to receive.

penumfmtetc
[in] Pointer to the **IEnumFORMATETC** interface that can be used to enumerate formats for format negotiation during binding. This parameter can be NULL, in which case the caller is not interested in format negotiation during binding, and the default format of the object will be bound to.

ppbc
[out] Indirect pointer to the **IBindCtx** interface for the new bind-context.

Return Values

S_OK
The operation completed successfully.

E_OUTOFMEMORY
The method ran out of memory and did not complete.

E_INVALIDARG
One or more parameters are invalid.

Remarks

This function automatically registers the IBindStatusCallback and the **IEnumFORMATETC** interfaces with the bind context. The *grfBSCOption* parameter allows the client to specify flags indicating which callback notifications the client is capable of receiving. If the client does not wish to receive certain notification, it can choose to implement those callback methods as empty function stubs (returning E_NOTIMPL), and they should not be called.

The **RegisterBindStatusCallback** function can also be used to register callback interfaces in the bind context.

See Also

IBindStatusCallback, BSCO_OPTION, RegisterBindStatusCallback

CreateFormatEnumerator

Creates an object that implements **IEnumFORMATETC** over a static array of **FORMATETC** structures.

```
HRESULT CreateFormatEnumerator(
    UINT   cfmtetc,                 //Number of FORMATETC structures in rgfmtetc
    FORMATETC *rgfmtetc,            //Static array of formats.
    IEnumFORMATETC **ppenumfmtetc   //Indirect pointer to enumerator object
  );
```

Parameters

cfmtetc
[in] Number of **FORMATETC** structures in the static array specified by the *rgfmtetc* parameter. The *cfmtetc* parameter cannot be zero.

rgfmtetc
[in] Pointer to a static array of **FORMATETC** structures.

ppenumfmtetc
[out] Indirect pointer to the **IEnumFORMATETC** interface on the enumerator object.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One or more parameters are invalid.

Remarks

The **CreateFormatEnumerator** function creates an enumerator object that implements **IEnumFORMATETC** over a static array of **FORMATETC** structures. The *cfmtetc* parameter specifies the number of these structures. With the pointer, you can call the standard enumeration methods to enumerate the structures, as described in the **IEnum*XXX*** reference.

See Also

RegisterFormatEnumerator, RevokeFormatEnumerator

CreateURLMoniker

Creates an URL moniker from either a full URL string or from a base context URL moniker and a partial URL string.

```
HRESULT CreateURLMoniker(
    IMoniker *pmkContext,      //Pointer to the base context moniker
    LPWSTR   szURL,            //Display name to be parsed
    IMoniker **ppmk            //Indirect pointer to IMoniker for the new URL moniker
  );
```

Parameters

pmkContext

[in] Pointer to the **IMoniker** interface for the URL moniker to use as the base context when the *szURL* parameter is a partial URL string. The *pmkContext* parameter can be NULL. In this case, one of the following methods is used to provide the context:

- The *szURL* parameter is a full URL string needing no further context.
- The leftmost portion of the moniker contains the URL context.
- The moniker retrieves the URL context from the bind-context during **IMoniker::BindToObject** or **IMoniker::BindToStorage**.

szURL

[in] Display name to be parsed.

ppmk

[out] Indirect pointer to an **IMoniker** interface for the new URL moniker.

Return Values

S_OK

The operation was successful.

E_OUTOFMEMORY

The operation ran out of memory.

MK_E_SYNTAX

A moniker could not be created because *szURL* does not correspond to valid URL syntax for a full or partial URL. This is uncommon, since most parsing of the URL occurs during binding and also since the syntax for URLs is extremely flexible.

Remarks

Partial URLs are similar to relative paths within file systems, in that resolution to an object requires a context outside the partial string alone. Full URL strings are like fully-qualified paths; they are self-contained and often location-independent.

When creating an URL moniker from a partial URL string specified in *szURL*, the caller can specify a context with a partial URL moniker in the *pmkContext* parameter. In this case, the **CreateURLMoniker** function retrieves the display name of *pmkContext* (by calling the **IMoniker::GetDisplayName** method) and manually composes it with *szURL* according to URL composition rules.

The caller can alternately create a moniker from a partial URL string when the *pmkContext* parameter is set to NULL. In this case, the resulting moniker obtains further context during binding (either through **IMoniker::BindToObject** or **IMoniker::BindToStorage**). The moniker goes through the following steps to obtain the additional context:

1. First, it obtains the URL context from the passed **IBindCtx** by using the following call:

```
IBindCtx::GetObjectParam(SZ_URLCONTEXT, (IUnknown**)&pmkContext)
```

2. Then, the moniker obtains the URL context from its leftmost portion which contains another URL moniker from which to obtain the URL context.

See Also

FindMediaTypeClass

Attempts to retrieve the CLSID for the specified media type.

```
HRESULT FindMediaTypeClass(
  LPBC pbc,              //Pointer to the bind context
  LPCSTR  szType,        //Media types
  CLSID  *pclsID,        //Pointer to the CLSID
  DWORD dwReserved       //Reserved for future use; must be zero
);
```

Parameters

pbc
[in] Pointer to the bind context on which the media type is registered.

szType
[in] String identifying the media types. This parameter cannot be NULL.

pclsID
[out] Pointer to the CLSID corresponding to the specified media types in *szType*.

dwReserved
[in] Reserved for future use; must be zero.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One or more parameters are invalid.

See Also

RegisterMediaTypeClass, RegisterMediaTypes

HlinkCreateBrowseContext

Creates an empty, default instance of the system browse context object.

HRESULT HlinkCreateBrowseContext(
  IUnknown ****punkOuter*,** //Controlling unknown for possible aggregation
  REFIID *riid***,** //Interface ID to return on the new browse context
  **void *****ppv* //Receives the requested interface's pointer
);

Parameters

punkOuter
[in] Pointer to the controlling **IUnknown** for the new browse context. Typically NULL, in which case the new browse context is not aggregated. This interface must be derived from **IUnknown**.

riid
[in] Identifies the interface to return on the new browse context. Typically IID_IHlinkBrowseContext, although it must be IID_IUnknown when *punkOuter* is non-NULL so that the aggregator can retrieve the new browse context's inner **IUnknown** for future delegation of **QueryInterface**. See the COM aggregation documentation for details.

ppv
[out] Location to return the *riid* interface.

Return Values

S_OK
The browse context was created and the interface pointer retrieved.

Remarks

This helper API is identical to calling

```
coCreateInstance(CLSID_StdHlinkBrowseContext, punkOuter,
    CLSCTX_SERVER, riid, ppv).
```

See Also

IHlinkBrowseContext, IHlinkFrame::GetBrowseContext, IHlinkTarget::GetBrowseContext

HlinkCreateFromData

Creates a standard hyperlink object from an OLE object that supports the **IDataObject** interface. Typically, this object originates from a data transfer operation, such as copy-paste using the clipboard, or via drag-and-drop.

HRESULT HlinkCreateFromData(

IDataObject* *pdatobj*,	//Interface on target data from which to create the hyperlink
IHlinkSite* *phlSite*,	//Interface on hyperlink site
DWORD *dwSiteData*,	//Additional site data
IUnknown **punkOuter*,	//Interface which controls aggregation
REFIID *riid*,	//Interface ID on new hyperlink object
**Void ** ** *ppv*	//Receives the requested interface's pointer

);

Parameters

pdatobj

[in] Pointer to the **IDataObject** interface pointer on the data object of interest. Used to access the methods to retrieve the data from the object.

phlSite

[in] Pointer to the IHlinkSite interface pointer for the new hyperlink within its container.

dwSiteData

[in] Additional site data associated with the hyperlink site.

punkOuter

[in] Pointer to the controlling **IUnknown** interface for the new hyperlink object. If NULL, the new hyperlink object is not aggregated.

riid

[in] Identifies the interface to return on the new hyperlink. Typically IID_IHlink, or IID_IUnknown when *punkOuter* is non-NULL.

ppv

[out] On return, pointer to the pointer to the requested interface.

Return Values

S_OK

The new hyperlink object was created successfully.

E_NOINTERFACE

The object did not support the *riid* interface.

Remarks

HlinkCreateFromData is one of three helper API's that you can use to create a hyperlink. **HlinkCreateFromData** creates a hyperlink from an OLE object which both supports the **IDataObject** interface and supports a hyperlink format CF_HYPERLINK on that **IDataObject** interface. **HlinkQueryCreateFromData** is typically called before the call to **HlinkCreateFromData** to determine if both these conditions are met. To create a hyperlink from an OLE object for which you know the moniker, use **HlinkCreateFromMoniker**. To create a hyperlink from an OLE object for which you have the name but not the moniker, use **HlinkCreateFromString**.

HlinkCreateFromData is typically used by a hyperlink container as part of a user interface that creates new hyperlinks via the clipboard or a drag-and-drop operation.

See Also
HlinkCreateFromMoniker, HlinkCreateFromString, HlinkQueryCreateFromData

HlinkCreateFromMoniker

Creates a new system hyperlink object from a moniker, a location string, and a friendly name string (used for displaying the hyperlink).

```
HlinkCreateFromMoniker(
   IMoniker * pmkTarget,          //Interface on hyperlink target moniker
   LPCWSTR szLocation,            //Named location of hyperlink reference within target
   LPCWSTR szFriendlyName,        //Display name of hyperlink
   IHlinkSite * phlSite,          //Interface on hyperlink site
   DWORD dwSiteData,              //Additional site data
   IUnknown * punkOuter,          //Whether or not object is part of an aggregate
   REFIID riid,                   //Interface ID on new hyperlink object
   Void * ppv                     //Receives the requested interface's pointer
  );
```

Parameters

pmkTarget
[in] A pointer to the moniker to the hyperlink target for the new hyperlink. May not be NULL.

szLocation
[in] A named location within the hyperlink target which resolves to the hyperlink reference.

szFriendlyName
[in] A string to use as the display name for the hyperlink.

phlSite
[in] Pointer to the IHlinkSite object for the new hyperlink within the hyperlink container.

dwSiteData
[in] Additional site data for the new hyperlink object.

punkOuter
[in] Pointer to the controlling **IUnknown** interface for the new hyperlink object. If NULL, the new hyperlink object is not aggregated.

riid
[in] Identifies the interface to return on the new hyperlink. Typically IID_IHlink, or IID_IUnknown when *punkOuter* is non-NULL.

ppv
[out] On return, pointer to the pointer to the requested interface.

Return Values

S_OK
A new hyperlink object was created successfully.

Remarks

HlinkCreateFromData is one of three helper API's that you can use to create a hyperlink. **HlinkCreateFromData** creates a hyperlink from an OLE object which both supports the **IDataObject** interface and supports a hyperlink format CF_HYPERLINK on that **IDataObject** interface. **HlinkQueryCreateFromData** is typically called before the call to **HlinkCreateFromData** to determine if both these conditions are met. To create a hyperlink from an OLE object for which you know the moniker, use **HlinkCreateFromMoniker**. To create a hyperlink from an OLE object for which you have the name but not the moniker, use **HlinkCreateFromString**.

HlinkCreateFromMoniker creates new hyperlinks from an existing target moniker, a location string within the target, and a friendly name string used to identify how to display the hyperlink. This method's typical use is as part of a user interface for creating new hyperlinks from existing hyperlinks, or for editing existing hyperlinks, using a dialog or form to prompt for the strings. This function is significantly faster than **HlinkCreateFromString**.

See Also

HlinkCreateFromData, HlinkCreateFromString, HlinkQueryCreateFromData

HlinkCreateFromString

Creates a new hyperlink object from strings representing the hyperlink target, the location within the target, and a friendly name.

```
HlinkCreateFromString(
   LPCWSTR szTarget,            //String helping to identify the hyperlink target
   LPCWSTR szLocation,          //Location within the hyperlink target of new hyperlink object
   LPCWSTR szFriendlyName,      //Friendly name of the hyperlink
   IHlinkSite * phlSite,        //Site for the new hyperlink object
   DWORD dwSiteData,            //Additional site data for the new hyperlink object
   IUnknown * punkOuter,        //Whether or not object is part of an aggregate
   REFIID riid,                 //Interface ID on new browse context object
   Void * ppv                   //Location of the requested interface's pointer
  );
```

Parameters

szTarget
[in] String to help identify the hyperlink target. This string is resolved into a moniker via **MkParseDisplayNameEx**.

szLocation
[in] The string representing the location within the hyperlink target for the new hyperlink. May not be NULL.

szFriendlyName
[in] The string to use as the friendly name for the hyperlink. This is typically a display name referenced by the user interface.

phlSite
[in] Pointer to the IHlinkSite object for the new hyperlink within the hyperlink container.

dwSiteData
[in] Additional site data for the new hyperlink object.

punkOuter
[in] Points to the controlling **IUnknown** for the new browse context. If NULL, the new browse context is not aggregated.

riid
[in] Identifies the interface to return on the new hyperlink. Typically IID_IHlink, or IID_IUnknown when *punkOuter* is non-NULL.

ppv
[out] On return, pointer to the pointer to the requested interface.

Return Values

S_OK
A new hyperlink object was created.

Remarks

HlinkCreateFromData is one of three helper API's that you can use to create a hyperlink. **HlinkCreateFromData** creates a hyperlink from an OLE object which both supports the **IDataObject** interface and supports a hyperlink format CF_HYPERLINK on that **IDataObject** interface. **HlinkQueryCreateFromData** is typically called before the call to **HlinkCreateFromData** to determine

if both these conditions are met. To create a hyperlink from an OLE object for which you know the moniker, use **HlinkCreateFromMoniker**. To create a hyperlink from an OLE object for which you have the name but not the moniker, use **HlinkCreateFromString**.

HlinkCreateFromString is typically used by hyperlink containers as part of the user-interface for creating brand new hyperlinks, where the user fills in a form or dialog of items -- target, location, friendly name -- which are used to construct the hyperlink.

See Also

HlinkCreateFromData, HlinkCreateFromMoniker, HlinkQueryCreateFromData

HlinkGetSpecialReference

For a given value from the HLSR enumeration, this function returns the current user's default global home, search, or history page for browsing as a string.

```
HlinkGetSpecialReference(
    DWORD dwReference,          //Special reference flags
    LPWSTR ** pszReference      //Buffer to return special reference string
  );
```

Parameters

dwReference

[in] A value taken from the **HLSR** enumeration.

pszReference

[out] Pointer to the location to return the special reference string. May not be NULL.

See Also

HlinkSetSpecialReference, HLSR

HlinkNavigate

This helper API can be used to navigate a hyperlink given a hyperlink object and an optional hyperlink frame object.

HlinkNavigate(

IHlink * *phl*,	//The hyperlink to navigate to
IHlinkFrame * *phlFrame*,	//The hyperlink frame of the hyperlink container.
DWORD *grfHLNF*,	//Navigation flags
IBindCtx * *pbc*,	//Bind context object interface pointer
IBindStatusCallback * *pbsc*,	//Bind status callback object interface pointer
IHlinkBrowseContext * *phlbc*	//Browse context object interface pointer

);

Parameters

phl

[in] Pointer to the IHlink interface on the hyperlink to navigate to.

phlFrame

[in] Pointer to the IHlinkFrame interface of the hyperlink container. May be NULL if the hyperlink container does not have a hyperlink frame.

grfHLNF

[in] Value taken from the HLNF enumeration.

pbc

[in] **IBindCtx** interface pointer to the bind context to use for any moniker binding performed during the navigation. May not be NULL.

pbsc

[in] IBindStatusCallback interface pointer to the bind status context to use for any asynchronous moniker binding performed during the navigation. May be NULL, in which case the caller is not interested in progress notification, cancellation, pausing, or low-level binding information.

phlbc

[in] Pointer to the IHlinkBrowseContext interface pointer to use for this navigation. The browse context includes history information in which this navigation is logged, if !(grfHLNF & HLNF_CREATENOHISTORY).

Remarks

HlinkNavigate encapsulates the following common sequence of calls:

```
if (phlFrame)
    phlFrame->Navigate(grfHLNF, pbc, pbsc, phl);
else if (phl)
    phl->Navigate(grfHLNF, pbc, pbsc, phlbc);
```

See Also

IHlinkBrowseContext, IHlink::Navigate, IHlinkFrame::Navigate, IHlinkTarget::Navigate

HlinkNavigateToStringReference

Creates a hyperlink site from strings representing the hyperlink target, the location within the target, and a friendly name, and then navigates to that site.

```
HlinkNavigateToStringReference(
    LPCWSTR szTarget,                   //String helping to identify the hyperlink target
    LPCWSTR szLocation,                 //Location within the hyperlink target of new hyperlink
    IHlinkSite * phlSite,               //Site object interface for the new hyperlink object
    DWORD dwSiteData,                   //Additional site data for the new hyperlink object
    IHlinkFrame * phlFrame,             //Frame object interface for the new hyperlink object
    DWORD grfHLNF,                      //Navigation flags
    IBindCtx * pbc,                     //Bind context object
    IBindStatusCallback * pbsc,         //Bind status callback object
    IHlinkBrowseContext * phlbc         //Browse context object
  );
```

Parameters

szTarget

[in] String which helps identify the hyperlink target. This string is resolved into a moniker for underlying binding operations via **MkParseDisplayNameEx**. May not be NULL.

szLocation

[in] Location within the hyperlink target of new hyperlink object.

phlSite

[in] Pointer to the IHlinkSite interface on the site object for the new hyperlink object. This is optional, in which case *szTarget* must be an absolute reference.

dwSiteData

[in] Additional site data for the new hyperlink object

phlFrame

[in] Pointer to IHlinkFrame interface for the hyperlink frame object of the hyperlink container object. May be NULL if the hyperlink container does not have a hyperlink frame.

grfHLNF

[in] Value taken from the HLNF enumeration.

pbc

[in] **IBindCtx** interface pointer to the bind context object to use for any moniker binding performed during the navigation. May not be NULL.

pbsc

[in] IBindStatusCallback interface pointer to the bind-status-callback object to use for any asynchronous moniker binding performed during the navigation. May be NULL, in which case the caller is not interested in progress notification, cancellation, pausing, or low-level binding information.

phlbc

[in] IHlinkBrowseContext interface pointer to the browse context object to use for this navigation. The browse context includes history information in which this navigation is logged, if !(*grfHLNF* & HLNF_CREATENOHISTORY).

Return Values

S_OK

A hyperlink site was created and navigated to.

Remarks

This helper API is identical to calling:

```
// create hyperlink site, IBindStatusCallback,
    // gather bind context, and browse context
HlinkCreateFromString(szTarget, szLocation,
    szFriendlyName, &hlSite, dwSiteData, NULL,
    IID_IHlink, (void**)&phl);
HlinkNavigate(phl, phlFrame, grfHLNF, pbc, pbsc, phlbc);
phl->Release();
```

See Also

HLNF, HlinkNavigate, HlinkCreateFromString, HlinkSimpleNavigateToString

HlinkOnNavigate

Notifies a hyperlink browse context and hyperlink frame, if it exists, that a hyperlink target has been navigated to.

```
HlinkOnNavigate(
    IHlinkFrame * phlFrame,          //Frame object interface for the new hyperlink
    IHlinkBrowseContext * phlbc,     //Browse context object to use for this navigation
    DWORD grfHLNF,                   //Navigation flags
    LPCWSTR szTarget,                //String helping to identify the hyperlink target
    LPCWSTR szLocation,              //Location within the hyperlink target of new hyperlink
    LPCWSTR szFriendlyName           //Friendly name of the hyperlink
  );
```

Parameters

phlFrame
[in] Pointer to the IHlinkFrame interface of the hyperlink container. May be NULL if the hyperlink container does not have a hyperlink frame.

phlbc
[in] Pointer to the IHlinkBrowseContext interface for the browse context object to use for this navigation. The browse context includes history information in which this navigation is logged, if ! (*grfHLNF* & HLNF_CREATENOHISTORY).

grfHLNF
[in] Value taken from the **HLNF** enumeration.

szTarget
[in] String which helps identify the hyperlink target. This string is resolved into a moniker for underlying binding operations via **MkParseDisplayNameEx**. May not be NULL.

szLocation
[in] Location within the hyperlink target of new hyperlink object.

szFriendlyName
[in] The friendly name of the hyperlink.

Return Values

S_OK
The browse context and the frame, if it exists, have been notified of the hyperlink navigation.

Remarks

HlinkOnNavigate is a helper API typically called during IHlinkTarget::Navigate which encapsulates the following calls:

```
phlbc->OnNavigateHlink(grfHLNF, pmkTarget, szLocation, szFriendlyName);
if (phlframe)
    phlframe->OnNavigate(grfHLNF);
```

See Also

IHlinkBrowseContext::OnNavigateHlink, IHlinkFrame::OnNavigate, IHlinkSite::OnNavigationComplete

HlinkQueryCreateFromData

Determines if a new hyperlink can be created from an **IDataObject**.

HRESULT HlinkQueryCreateFromData(
  IDataObject* *pdatobj* //Data object to query
);

Parameters

pdatobj
  [in] Pointer to the **IDataObject** supplying the hyperlink formats.

Return Values

S_OK
  A hyperlink can be created from the data object.

S_FALSE
  A hyperlink can not be created from the data object.

Remarks

A hyperlink can be created from an **IDataObject** in two ways:

- The IDataObject offers the clip format CF_HYPERLINK on storage media type TYMED_ISTREAM or TYMED_HGLOBAL, enumerations on **TYMED**
- The IDataObject offers Win95 shortcut data

See Also

HlinkCreateFromData

HlinkSetSpecialReference

For a given value from the HLSR enumeration, this function sets the current user's default global home, search, or history page for browsing as a string.

```
HlinkSetSpecialReference(
    DWORD dwReference,      //Special reference flags
    LPCWSTR szReference     //The special reference string
  );
```

Parameters

dwReference
[in] A value taken from the **HLSR** enumeration.

szReference
[in] The special reference string.

See Also

HlinkGetSpecialReference, HLSR

HlinkSimpleNavigateToMoniker

This helper function should be used by all application, documents, and objects that have simple navigation needs. This single function call that will “do the right thing” depending on whether the navigation is originating from within a hyperlink frame or within a hyperlink-unaware application.

```
HlinkSimpleNavigateToMoniker(
   IMoniker * pmkTarget,                //Moniker of the hyperlink target
   LPCWSTR szLocation,                  //Optional string representing location within target
   LPCWSTR szTargetFrameName,           //Optional string naming the target frame
   LPCWSTR szAdditionalParams,          //Optional additional parameters
   IUnknown * punk,                     //IUnknown interface of initiating document or object
   DWORD grfHLNF,                       //Navigation flags
   IBindCtx * pbc,                      //Bind context object
   IBindStatusCallback * pbsc,          //Bind status callback
   DWORD dwReserved                     //Reserved for future use
  );
```

Parameters

szTarget

[in] String which helps identify the hyperlink target. This string is resolved into a moniker for underlying binding operations via MkParseDisplayNameEx. May not be NULL.

szLocation

[in] Optional string representing the location within the hyperlink target for the new hyperlink.

szTargetFrameName

[in] Optional string naming the target frame for the hyperlink navigation. This argument only affects navigation within a document container that understands frame-sets.

szAdditionalParams

[in] Additional string parameters for the hyperlink navigation (currently ignored). IHlink::SetAdditionalParams and IHlink::GetAdditionalParams describe the format.

pUnk

[in] The **IUnknown** pointer to the document or object that is initiating the hyperlink. If NULL, it is assumed the hyperlink originates from an OLE-unaware application.

grfHLNF

[in] Value taken from the HLNF enumeration.

pbc

[in] Bind context to use for any moniker binding performed during the navigation. May not be NULL.

pbsc

[in] Bind-status-callback to use for any asynchronous moniker binding performed during the navigation. May be NULL, in which case the caller is not interested in progress notification, cancellation, pausing, or low-level binding information.

dwReserved

[in] Reserved for future use, must be set to NULL.

Return Values

S_OK

A hyperlink knowing the moniker of its target has navigated to it successfully.

This function can also return any value returned by IHlink::Navigate.

See Also

HlinkNavigate, HlinkNavigateToStringReference, HlinkSimpleNavigateToString, HLNF

HlinkSimpleNavigateToString

This helper function should be used by all application, documents, and objects that have simple navigation needs. This single function call that will “do the right thing” depending on whether the navigation is originating from within a hyperlink frame or within a hyperlink-unaware application.

HlinkSimpleNavigateToString(

LPCWSTR *szTarget*,	//String to be resolved into target’s moniker
LPCWSTR *szLocation*,	//Optional string representing location within target
LPCWSTR *szTargetFrameName*,	//Optional string naming the target frame
LPCWSTR *szAdditionalParams*,	//Additional parameters
IUnknown * *pUnk*,	//**IUnknown** interface of initiating document or object
DWORD *grfHLNF*,	//Navigation flags
IBindCtx * *pbc*,	//Bind context object
IBindStatusCallback * *pbsc*,	//Bind status callback object
DWORD *dwReserved*	//Reserved for future use

);

Parameters

szTarget

[in] String which helps identify the hyperlink target. This string is resolved into a moniker for underlying binding operations via **MkParseDisplayNameEx**. If NULL, then the navigation is within a document. This parameter is required.

szLocation

[in] Optional string representing the location within the hyperlink target for the new hyperlink.

szTargetFrameName

[in] Optional string naming the target frame for the hyperlink navigation. This argument only affects navigation within a document container that understands frame-sets.

szAdditionalParams

[in] Additional string parameters for the hyperlink navigation (currently ignored). **IHlink::SetAdditionalParams** and **IHlink::GetAdditionalParams** describe the format.

pUnk

[in] The **IUnknown** pointer to the document or object that is initiating the hyperlink. If NULL, it is assumed the hyperlink originates from an OLE-unaware application.

grfHLNF

[in] Value taken from the HLNF enumeration.

pbc

[in] **IBindCtx** interface pointer to the bind context object to use for any moniker binding performed during the navigation. May not be NULL.

pbsc

[in] **IBindStatusCallback** interface pointer to the bind-status-callback object to use for any asynchronous moniker binding performed during the navigation. May be NULL, in which case the caller is not interested in progress notification, cancellation, pausing, or low-level binding information.

dwReserved

[in] Reserved for future use, must be set to NULL.

Return Values

S_OK
A hyperlink knowing the name of its target has navigated to it successfully.

This function can also return any value returned by IHlink::Navigate.

See Also

HlinkNavigate, HlinkNavigateToStringReference, HlinkSimpleNavigateToMoniker, HLNF, IHlink::Navigate

IsAsyncMoniker

Tests if a moniker supports asynchronous binding.

HRESULT IsAsyncMoniker(
  IMoniker ******pmk* //Pointer to the IMoniker interface on the moniker to be tested
);

Parameters

pmk
  [in] Pointer to the **IMoniker** interface on the moniker to be tested.

Return Values

S_OK
  The specified moniker is asychronous.

E_INVALIDARG
  The *pmk* parameter is invalid.

Remarks

A moniker implementation indicates that it is asynchronous by supporting the **IAsyncMoniker** interface, an empty interface that is just an implementation of **IUnknown**. The **IsAsyncMoniker** function tests for support of **IAsyncMoniker** and also handles composite monikers correctly.

See Also

IAsyncMoniker

MkParseDisplayNameEx

Given a string, this function returns a moniker of the object that the string denotes.

```
HRESULT MkParseDisplayNameEx(
    IBindCtx* pbc,              //Pointer to the bind context
    LPWSTR    szDisplayName,    //Display name to be parsed
    ULONG *   pcchEaten,        //Pointer to the number of characters of the display name successfully parsed
    IMoniker ** ppmk            //Indirect pointer to a moniker
  );
```

Parameters

pbc
[in] Pointer to the bind context in which to accumulate bound objects.

szDisplayName
[in] Display name to be parsed.

pcchEaten
[out] Pointer to the number of characters of the display name that were successfully parsed. Most useful on syntax error, when a non-zero value is often returned and therefore a subsequent call to **MkParseDisplayNameEx** with the same *pbc* and a shortened *szDisplayName* should return a valid moniker.

ppmk
[out] Indirect pointer to the resulting moniker.

Return Values

S_OK
The operation was successful.

MK_E_SYNTAX
Parsing failed because *szDisplayName* could only be partially resolved into a moniker. In this case, **pcchEaten* has the number of characters that were successfully parsed into a moniker prefix.

E_OUTOFMEMORY
The operation ran out of memory.

Remarks

Given a string, this function returns a moniker for the object that the string denotes. This operation is known as parsing. A display name is parsed into a moniker; it is resolved into its component moniker parts.

If a syntax error occurs, then an indication of how much of the string was successfully parsed is returned in **pcchEaten* and NULL is returned through ***ppmk*. Otherwise, the value returned through **pcchEaten* indicates the entire size of the display name.

This function differs from the original **MkParseDisplayName** function in that it supports Universal Resource Indicator (URI) syntax. See the IETC RFC1630 specification for more information on URIs.

Parsing a display name may in some cases be as expensive as binding to the object that it denotes, since, along the way, the parsing mechanism must connect to various non-trivial name space managers (such as a spreadsheet application that can parse into ranges in its sheets). As might be expected, objects are not released by the parsing operation itself, but are instead handed over to the bind context that was passed in through **IBindCtx::RegisterObjectBound**. Thus, if the moniker

resulting from the parse is immediately bound using this same bind context, redundant loading of objects is maximally avoided.

In many other cases, however, parsing a display name may be quite inexpensive since a single name-space manager may quickly return a moniker that will perform further expensive analysis on any acceptable name during **IMoniker::BindToObject** or other methods. An example of such an inexpensive parser is the Win32 implementation of a File Moniker. A theoretical example would be a naive URL moniker which parsed from any valid URL strings (i.e., “http:…”, “file:…”) and only during binding took time to resolve the string against the Internet, a potentially expensive operation.

The parsing process is an inductive one, in that there is an initial step that gets the process going, followed by the repeated application of an inductive step. At any point after the beginning of the parse, a certain prefix of *szDisplayName* has been parsed into a moniker, and a suffix of the display name remains unresolved.

The inductive step asks the moniker-so-far using **IMoniker::ParseDisplayName** to consume as much as it would like of the remaining suffix and return the corresponding moniker and the new suffix. The moniker is composed onto the end of the existing moniker, and the process repeats.

Implementations of **IMoniker::ParseDisplayName** vary in exactly where the knowledge of how to carry out the parsing is kept. Some monikers by their nature are only used in particular kinds of containers. It is likely that these monikers themselves have the knowledge of the legal display name syntax within the objects that they themselves denote, so they can carry out the processes completely within **IMoniker::ParseDisplayName**. The common case, however, is that the moniker is generic in the sense that is not specific to one kind of container, and thus cannot know the legal syntax for elements within the container. File monikers are an example of these, as are Item Monikers. These monikers in general employ the following strategy to carry out parsing. First, the moniker connects to the class of object that it currently denotes, asking for **IParseDisplayName** interface. If that succeeds, then it uses the obtained interface pointer to attempt to carry out the parse. If the class refuses to handle the parse, then the moniker binds to the object it denotes, asking again for **IParseDisplayName** interface. If this fails, then the parse is aborted.

The effect is that ultimately an object always gets to be in control of the syntax of elements contained inside of itself. It’s just that objects of a certain nature can carry out parsing more efficiently by having a moniker or their class do the parsing on their behalf.

Notice that since **MkParseDisplayNameEx** knows nothing of the legal syntax of display names (with the exception of the initial parsing step; see below). It is of course beneficial to the user that display names in different contexts not have gratuitously different syntax. While there some rare situations which call for special purpose syntax, it is recommended that, unless there are compelling reasons to do otherwise, the syntax for display names should be the same as or similar to the native file system syntax. The aim is to build on user familiarity. Most important about this are the characters allowed for the delimiters used to separate the display name of one of the component monikers from the next. Unless through some special circumstances they have very good reason not to, all moniker implementations should use inter-moniker delimiters from the following character set:

\ / : ! [

Standardization in delimiters promotes usability. But more importantly, notice that the parsing algorithm has the characteristic that a given container consumes as much as it can of the string being parsed before passing the remainder on to the designated object inside themselves. If the delimiter expected of the next-to-be-generated moniker in fact forms (part of) a valid display name in the container, then the container’s parse will consume it!

Monikers and objects which have implementations on more than one platform (such as File Monikers) should always parse according to the syntax of the platform on which they are currently running. When asked for their display name, monikers should also show delimiters appropriate to the platform on which they are currently running, even if they were originally created on a different platform. In total,

users will always deal with delimiters appropriate for the host platform.

The initial step of the parsing process is a bit tricky, in that it needs to somehow determine the initial moniker. **MkParseDisplayNameEx** is omniscient with respect to the syntax with which the display name of a moniker may legally begin, and it uses this omniscience to choose the initial moniker.

The initial moniker is determined by trying the following strategies in order, using the first to succeed.

1. "**ProgID:" Case:** If a prefix of *szDisplayName* conforms to the legal ProgID syntax, is more than 1 character long, and is followed by a colon (':'), the ProgID is converted to a CLSID with **CLSIDFromProgID***,* An instance of this class is asked for the **IParseDisplayName** interface, and **IParseDisplayName:ParseDisplayName** is called with the entire *szDisplayName.* This case distinguishes **MkParseDisplayNameEx** from **MkParseDisplayName**.
2. **ROT Case:** All prefixes of *szDisplayName* that consist solely of valid file name characters are consulted as file monikers in the Running Object Table.
3. **File-System Case:** The file system is consulted to check if a prefix of *szDisplayName* matches an existing file. Said file name may be drive absolute, drive relative, working-directory relative, or begin with an explicit network share name. This is a common case.
4. **"@ProgID" Case:** If the initial character of *szDisplayName* is '@', then the maximal string immediately following the '@' which conforms to the legal ProgID syntax is determined. This is converted to a CLSID with **CLSIDFromProgID**. An instance of this class is asked in turn for **IParseDisplayName** interface; the **IParseDisplayName** interface so found is then given the whole string (starting with the '@') to continue parsing.

See Also

IParseDisplayName

RegisterBindStatusCallback

Registers a callback interface with an existing bind context.

```
HRESULT RegisterBindStatusCallback(
  IBindCtx *pbc,                  //Pointer to the bind context in which to register the callback
  IBindStatusCallback *pbsc,      //Pointer to callback interface to be registered
  DWORD    grfBSCOption,          //BSCO_OPTION flags
  DWORD    dwReserved             //Reserved for future use
);
```

Parameters

pbc
[in] Pointer to the **IBindCtx** interface for the bind context in which the callback is to be registered.

pbsc
[in] Pointer to the **IBindStatusCallback** interface to be registered.

grfBSCOption
[in] Flags from the BSCO_OPTION enumeration specifying which callback methods should be called.

dwReserved
[in] Reserved for future use.

Return Values

S_OK
The operation was successful.

E_OUTOFMEMORY
There was insufficient memory to register the callback with the bind context.

E_INVALIDARG
One or more parameters are invalid.

Remarks

The callback interface that is registered with this function is used for all asynchronous bind operations involving the specified bind context until the callback interface is revoked through **RevokeBindStatusCallback** or until the bind context is destroyed.

RegisterBindStatusCallback allows the client to specify flags that determine the callback notifications that the client is capable of receiving. If the client does not wish to receive certain notifications, it can implement those callback methods as empty function stubs (returning E_NOTIMPL), and they will not be called.

RegisterBindStatusCallback allows multiple clients to each register a callback for the same bind context. During the bind operation, these callbacks are called in an arbitrary order, and the asynchronous moniker can set a policy and limit certain callback notifications to only one of the registered callbacks. For example, the notifications **GetBindInfo**, **OnDataAvailable**, or **OnObjectAvailable** are usually limited to a single registered callback interface. For this reason, a client should request to receive only those callback methods that it plans on implementing.

Specifically, even though multiple **IBindStatusCallback** interfaces can be registered on the bind context, only one moniker client will actually receive the following notifications: **GetBindInfo**, **OnDataAvailable**, or **OnObjectAvailable**. This client is the last one to register itself as capable of receiving these callbacks, and it is typically the client that calls **IMoniker::BindToStorage** or

IMoniker::BindToObject and drives the bind operation.

In the current implementation of URL monikers, the clients are called in reverse of the order in which they were registered, with the exception of the last client to request the following notifications: **GetBindInfo**, **OnDataAvailable**, or **OnObjectAvailable**. The last client in the callback order is notified for these exceptions. However, it is recommended that you not depend heavily on this ordering for the URL moniker implementation. The only important point is that the registered client for **GetBindInfo** or **OnDataAvailable**, or **OnObjectAvailable** is last in the ordering.

See Also

IBindStatusCallback, RevokeBindStatusCallback

RegisterFormatEnumerator

Registers a **FORMATETC** enumerator object onto the given bind context.

```
HRESULT RegisterFormatEnumerator(
   LPBC pbc,                        //Pointer to the bind context
   IEnumFORMATETC *pEFetc,          //Pointer to the IEnumFORMATETC interface
   DWORD  dwReserved                //Reserved for future use; must be zero
  );
```

Parameters

pbc
   [in] Pointer to the **IBindCtx** interface for the bind context on which to register the enumerator.

pEFetc
   [in] Pointer to the **IEnumFORMATETC** interface for the enumerator to be registered.

dwReserved
   [in] Reserved for future use; must be zero.

Return Values

S_OK
   The operation was successful.

E_INVALIDARG
   One or more parameters are invalid.

Remarks

The enumerator is used to determine what format types are prefered for the bind operation. Typically, the *pEFetc* parameter would be the pointer obtained through a call to **CreateFormatEnumerator**.

See Also

CreateFormatEnumerator, RevokeFormatEnumerator

RegisterMediaTypeClass

Registers a mapping of media types to CLSIDs to override the default mapping specified in the registry. The new mapping is used calls to **IMoniker::BindToObject** when binding objects in the specified bind context.

```
HRESULT RegisterMediaTypeClass(
    LPBC pbc,                //Pointer to the bind context
    UINT ctypes,             //Number of media type strings in rgszTypes
    LPCSTR *rgszTypes,       //Pointer to array of media types to be registered
    CLSID *rgclsID,          //Pointer to array of CLSIDs corresponding to rgszTypes
    DWORD dwReserved         //Reserved for future use; must be zero
  );
```

Parameters

pbc
[in] Pointer to the **IBindCtx** interface for the bind context in which the media types are to be registered.

ctypes
[in] Number of media type strings in the *rgszTypes* array. This parameter cannot be zero.

rgszTypes
[in] Pointer to an array of strings identifying the media types to be registered. None of these strings can be NULL.

rgclsID
[in] Pointer to an array of CLSIDs corresponding to the media type strings in the *rgszTypes* array.

dwReserved
[in] Reserved for future use; must be zero.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One or more parameters are invalid.

Remarks

This function is primarily used by moniker clients calling **IMoniker::BindToObject** to override the default registry mapping between mime types and CLSIDs. In most cases, the default mapping provided in the registry is used. However, a web browser may want the CLSID for its HTML viewer to be associated with .TXT files without changing the default registry association for text files. The override is used for all bind operations using the specified bind context.

See Also

RegisterMediaTypes

RegisterMediaTypes

Registers media types strings.

```
HRESULT RegisterMediaTypes (
    UINT    ctypes,             //Number of media type strings in rgszTypes
    LPTSTR *rgszTypes,          //Pointer to array of media types to be registered
    CLIPFORMAT *rgcfTypes       //Pointer to array of 32-bit values corresponding to rgszType array values
  );
```

Parameters

ctypes
[in] Number of media type strings in the *rgszTypes* array. This parameter cannot be zero.

rgszTypes
[in] Pointer to an array of strings identifying the media types to be registered. None of the strings in the array can be NULL.

rgcfTypes
[out] Pointer to an array of the 32-bit values assigned to corresponding media types in *rgszTypes*.

Return Values

S_OK
The operation was successful.

E_INVALIDARG
One or more parameters are invalid.

Remarks

Media types are taken from the following:

Value	Meaning
CF_NULL	0
SZ_URLCONTEXT	(L”URL Context”)
CFSTR_MIME_FRACTALS	(TEXT(“application/fractals”))
CFSTR_MIME_RAWDATA	(TEXT(“application/octet”))
CFSTR_MIME_POSTSCRIPT	(TEXT(“application/postscript”))
CFSTR_MIME_AIIF	(TEXT(“audio/aiff”))
CFSTR_MIME_BASICAUDIO	(TEXT(“audio/basic”))
CFSTR_MIME_WAV	(TEXT(“audio/wav”))
CFSTR_MIME_X_AIIF	(TEXT(“audio/x-aiif”))
CFSTR_MIME_X_REALAUDIO	(TEXT(“audio/x-pn-realaudio”))
CFSTR_MIME_X_WAV	(TEXT(“audio/x-wav”))
CFSTR_MIME_BMP	(TEXT(“image/bmp”))
CFSTR_MIME_GIF	(TEXT(“image/gif”))
CFSTR_MIME_JPEG	(TEXT(“image/jpeg”))
CFSTR_MIME_TIFF	(TEXT(“image/tiff”))
CFSTR_MIME_XBM	(TEXT(“image/xbm”))

CFSTR_MIME_X_BITMAP	(TEXT(“image/x-bitmap”))
CFSTR_MIME_HTML	(TEXT(“text/html”))
CFSTR_MIME_TEXT	(TEXT(“text/plain”))
CFSTR_MIME_AVI	(TEXT(“video/avi”))
CFSTR_MIME_MPEG	(TEXT(“video/mpeg”))
CFSTR_MIME_QUICKTIME	TEXT(“video/quicktime”))
CFSTR_MIME_X_MSVIDEO	(TEXT(“video/x-msvideo”))
CFSTR_MIME_X_SGI_MOVIE	(TEXT(“video/x-sgi-movie”))

See Also

RegisterMediaTypeClass

RevokeBindStatusCallback

Revokes a bind status callback interface previously registered on a bind context.

```
HRESULT RevokeBindStatusCallback(
    IBindCtx *pbc,                  //Pointer to the bind context from which the callback is to be revoked
    IBindStatusCallback *pbsc       //Pointer to the callback interface to revoke
  );
```

Parameters

pbc

[in] Pointer to the **IBindCtx** interface for the bind context from which the callback interface is to be revoked.

pbsc

[in] Pointer to the **IBindStatusCallback** interface to revoke.

Return Values

S_OK

The operation was successful.

E_FAIL

The callback interface specified is not registered on the specified bind context.

E_INVALIDARG

One or more parameters are invalid.

Remarks

RevokeBindStatusCallback will not succeed if it is made during a bind operation.

Note that it is not necessary to make this call for every use of a bind context. It is possible (although not recommended) to reuse the same bind context and the same callback for several bind operations. Upon calling **IBindCtx::Release**, all registered objects on that bind context are revoked, including the callback interfaces. Therefore, releasing a bind context implicitly releases all registered callbacks. However, if one chooses to reuse a bind context, one can use **RevokeBindStatusCallback** to remove a registered callback so it is not re-used.

See Also

IBindStatusCallback, RegisterBindStatusCallback

RevokeFormatEnumerator

Removes a format enumerator from the given bind context.

```
HRESULT RevokeFormatEnumerator(
    LPBC pbc,                       //Pointer to the bind context
    IEnumFORMATETC *pEFetc          //Pointer to the format enumerator to revoke
  );
```

Parameters

pbc

[in] Pointer to the **IBindCtx** interface for the bind context from which the enumerator is to be revoked.

pbsc

[in] Pointer to the **IEnumFORMATETC** interface for the enumerator to revoke.

Return Values

S_OK

The enumerator was successfully removed..

E_INVALIDARG

One or more parameters are invalid.

Remarks

RevokeFormatEnumerator removes a format enumerator from the bind context specified in *pbc*. It must previously have been registered with a call to **RegisterFormatEnumerator**.

Note that it is not necessary to make this call for every use of a bind context. It is possible (although not recommended) to reuse the same bind context and the same format enumerator for several bind operations. Upon calling **IBindCtx::Release**, all registered objects on that bind context are revoked, including the format enumerator interfaces. Therefore, releasing a bind context implicitly releases all registered format enumerators. However, if one chooses to reuse a bind context, one can use **RevokeFormatEnumerator** to remove a registered format enumerator so it is not re-used.

See Also

RegisterFormatEnumerator

BINDINFO

The client of an asynchronous moniker provides the **BINDINFO** structure and returns it to the asynchronous moniker when the moniker calls the client's IBindStatusCallback::GetBindInfo method. The **BINDINFO** structure provides additional information for the requested binding operation. The meaning of this structure is specific to the type of asynchronous moniker. The technical specification provided here describes the meaning of the structure when used for URL monikers.

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

Members

cbSize
Size of the structure, in bytes.

szExtraInfo
The behavior of this field is moniker specific. For URL monikers, this string is appended to the URL when the bind operation is started. Like other OLE strings, this value is a Unicode string that the client should allocate using **CoMemAlloc**. The URL Moniker will free the memory later.

stgmedData
Data to be used in a PUT or POST operation specified by the *dwBindVerb* member.

grfBindInfoF
Flag from the **BINDINFOF** enumeration that determines the use of URL encoding during the binding operation. This member is specific to URL monikers.

dwBindVerb
A value from the **BINDVERB** enumeration specifying an action to be performed during the bind operation.

szCustomVerb
String specifying a protocol specific custom action to be performed during the bind operation (only if *dwBindVerb* is set to BINDVERB_CUSTOM).

cbStgmedData
Size of the data provided in the *stgmedData* member.

See Also

BINDINFOF, BINDVERB, IBindStatusCallback::GetBindInfo

HLBWINFO

The **HLBWINFO** structure contains parameters relating to the locations and sizes of frame- and document-level windows within a browse context. The **HLBWINFO** structure is retrieved from the browse context using IHlinkBrowseContext::GetBrowseWindowInfo, and put into the browse context using IHlinkBrowseContext::SetBrowseWindowInfo. Hyperlink targets retrieve the **HLBWINFO** structure during IHlinkTarget::Navigate in order to reposition their user interface properly and ensure as seamless a transition as possible to the new document or object.

```
typedef struct tagHLBWINFO
{
    ULONG cbSize;
    DWORD grfHLBWIF;
    RECTL rcFramePos;
    RECTL rcDocPos;
} HLBWINFO;
```

Members

cbSize
Total size of this structure in bytes.

grfHLBWIF
Values taken from the HLBWIF enumeration.

rcFramePos
If *grfHLBWIF* & HLBWIF_HASFRAMEWNDINFO, contains the rectangle in screen coordinates of current frame-level windows within the browse context. When *grfHLBWIF* & HLBWIF_FRAMEWNDMAXIMIZED, frame-level windows are currently being displayed maximized. In this case *rcFramePos* is the "normal" size of frame-level windows, i.e. the rectangle to use for any frame-level window when it is non-maximized.

rcDocPos
If *grfHLBWIF* & HLBWIF_HASDOCWNDINFO, contains the rectangle in screen coordinates of current document-level windows within the browse context. When *grfHLBWIF* & HLBWIF_DOCWNDMAXIMIZED, document-level windows are currently being displayed maximized. In this case *rcDocPos* is the "normal" size of document-level windows, i.e. the rectangle to use for any document-level window when it is non-maximized.

See Also

HLBWIF, IHlinkBrowseContext::GetBrowseWindowInfo, IHlinkBrowseContext::SetBrowseWindowInfo, IHlinkTarget::Navigate

HLITEM

The **HLITEM** structure is the data structure used by the system browse context to track hyperlinks. This structure is returned by the IEnumHLITEM enumerator, which is returned from IHlinkBrowseContext::EnumNavigationStack.

```
typedef struct tagHLITEM
{
    ULONG uHLID;
    LPWSTR szFriendlyName;
} HLITEM;
```

Members

uHLID

Identifies the hyperlink. Standard enumerators never return one of the logical **HLID** constants in this field, always an identifier.

szFriendlyName

Friendly name of the hyperlink. Appropriate for display in the user interface.

See Also

HLID, IEnumHLITEM, IHlinkBrowseContext::EnumNavigationStack

OLECMD

The OLECMD strcuture is used to associate command flags from the OLECMDF enumeration with a command identifer through IOleCommandTarget::QueryStatus.

```
typedef struct _tagOLECMD
{
    ULONG cmdID;
    DWORD cmdf;
}OLECMDF;
```

Members

cmdID

A command identifier; taken from the OLECMDID enumeration..

cmdf

Flags associated with *cmdID;* taken from the OLECMDF enumeration.

See Also

OLECMDF, IOleCommandTarget::QueryStatus

OLECMDTEXT

Returns a text name or status string for a single command identifier. Used with IOleCommandTarget::QueryStatus.

```
typedef struct _tagOLECMDTEXT
{
    DWORD cmdtextf;
    ULONG cwactual;
    ULONG cwbuf;
    wchar_t rgwz[1];
}OLECMDTEXTF;
```

Members

cmdtextf
Filled on input; a value from the OLECMDTEXTF enumeration describing the information the caller wishes to receive in return.

cwActual
Filled on output; the number of characters actually written into the *rgwz* buffer before **QueryStatus** returns.

cwBuf
Filled on input; the size of the string buffer in *cwBuf*.

rgwz
A caller-allocated array of wide characters to receive the string on output.

See Also

IOleCommandTarget::QueryStatus, OLECMDTEXTF

PAGERANGE

Specifies a range of pages that begins and ends with page numbers that are selected by an end-user. The *rgPageRange* member of the PAGESET structure is a structure of type **PAGERANGE**.

```
typedef struct tagPAGERANGE
{
    LONG nFromPage;
    LONG nToPage;
}PAGERANGE;
```

Members

nFromPage

The first page to print. This member can have any page number as a value.If this value is greater than the value specified in *nToPage*, the document will be printed in reverse page order.

nToPage

The last page to print. A special value of PAGESET_TOLASTPAGE indicates that all the remaining pages should be printed. This member can have any page number as a value.If this value is less than the value specified in *nFromPage*, the document will be printed in reverse page order.

See Also

PAGESET

PAGESET

Identifies one or more page-ranges to be printed and, optionally, identifies only the even or odd pages as part of a pageset.

```
typedef struct tagPAGESET
{
    ULONG cbStruct;
    BOOL fOddPages;
    BOOL fEvenPages;
    ULONG cPageRange;
    [size_is(cPageRange)] PAGERANGE rgPages[]
}PAGESET;
```

Members

cbStruct
The number of bytes in this instance of the **PAGESET** structure. Must be a multiple of 4.

fOddPages
If true, then only the odd-numbered pages in the page-set indicated by *rgPages* are to be printed.

fEvenPages
If true, then only the even-numbered pages in the page-set indicated by *rgPages* are to be printed.

cPageRange
The number of page-range pairs specified in *rgPages*.

rgPages
Pointer to a PAGERANGE structure specifying the pages to be printed. One or more page ranges can be specified, so long as that number is the value of *cPageRange*. The page ranges must be sorted in increasing order and non-overlapping. It is an error to attempt to print a page which does not exist.

See Also

PAGERANGE

BINDF

Values from the BINDF enumeration are returned to the moniker from the client's IBindStatusCallback::OnStartBinding method. These values specify the type of binding the client wants from the moniker.

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

Elements

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. The client will be notificed through calls to its IBindStatusCallback::OnDataAvailable or IBindStatusCallback::OnObjectAvailable methods. If the client does not specify this flag, the bind operation will be synchronous, and the client will not receive any data from the bind operation until the **IMoniker::BindTo*Xxx*** call returns.

BINDF_ASYNCSTORAGE

The client of **IMoniker::BindToStorage** prefers that the storage and stream objects returned in **IBindStatusCallback::OnDataAvailable** return E_PENDING when they reference data not yet available through their read methods, rather than blocking until the data becomes available. This flag applies only to BINDF_ASYNCHRONOUS operations. Note that asynchronous stream objects return E_PENDING while data is still downloading and return S_FALSE for the end of the file.

BINDF_PULLDATA

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 drive the operation by pushing the data to the client. Specifically, when this flag is specified, 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 specifies 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 bind operation should retrieve the newest version of the data/object possible. For URL monikers, this flag maps to an HTTP IF MODIFIED SINCE request. Cached data is only used if it is the most recent version.

BINDF_NOWRITECACHE

The bind operation should not store retrieved data in the disk cache.

See Also

IBindStatusCallback::OnDataAvailable, IBindStatusCallback::OnObjectAvailable, IBindStatusCallback::OnStartBinding

BINDINFOF

Values from the BINDINFOF enumeration are passed to the moniker as part of the BINDINFO structure. The moniker calls the IBindStatusCallback::GetBindInfo method on the client to obtain additional information about the bind operation in the **BINDINFO** structure. The *grfBindInfof* member of the **BINDINFO** structure determines the use of URL encoding during the binding operation.

```
typedef enum tagBINDINFOF {
    BINDINFOF_URLENCODESTGMEDDATA,
    BINDINFOF_URLENCODEDEXTRAINFO
} BINDINFOF;
```

Elements

BINDINFOF_URLENCODESTGMEDDATA
: Use URL encoding to pass in the data provided in the *stgmedData* member of the BINDINFO structure for PUT and POST operations.

BINDINFOF_URLENCODEEXTRAINFO
: Use URL encoding to pass in the data provided in the *szExtraInfo* member of the BINDINFO structure.

See Also

IBindStatusCallback::GetBindInfo, BINDINFO

BINDSTATUS

A single value from the **BINDSTATUS** enumeration is passed to the client in the IBindStatusCallback::OnProgress method to indicate the progress of the bind operation.

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

Elements

BINDSTATUS_FINDINGRESOURCE
: The bind operation is finding the resource that holds the object or storage being bound to. The *szStatusText* parameter to the **IBindStatusCallback::OnProgress** method provides the display name of the resource being searched for (for example, “www.microsoft.com”).

BINDSTATUS_CONNECTING
: The bind operation is connecting to the resource that holds the object or storage being bound to. The *szStatusText* parameter to the **IBindStatusCallback::OnProgress** method provides the display name of the resource being connected to (for example, an IP address).

BINDSTATUS_SENDINGREQUEST
: The bind operation is requesting the object or storage being bound to. The *szStatusText* parameter to the **IBindStatusCallback::OnProgress** method provides the display name of the object (for example, a filename).

BINDSTATUS_REDIRECTING
: The bind operation has been redirected to a different data location. The *szStatusText* parameter to the **IBindStatusCallback::OnProgress** method 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* parameter to the **IBindStatusCallback::OnProgress** method is NULL.

BINDSTATUS_BEGINDOWNLOADDATA
: The bind operation has begun receiving the object or storage being bound to. The *szStatusText* parameter to the **IBindStatusCallback::OnProgress** method 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* parameter to the **IBindStatusCallback::OnProgress** method 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* parameter to the **IBindStatusCallback::OnProgress** method provides the display name of the data location.

See Also

IBindStatusCallback::OnProgress

BINDVERB

Values from the **BINDVERB** enumeration are passed to the moniker as part of the BINDINFO structure. The moniker calls the IBindStatusCallback::GetBindInfo method on the client to obtain additional information about the bind operation in the **BINDINFO** structure. The *dwBindVerb* member of the **BINDINFO** structure specifies an action, such as an HTTP request, to be performed during the binding operation.

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

Elements

BINDVERB_GET
: Perform an HTTP GET operation, the default operation. The *stgmedData* member of the BINDINFO structure should be set to TYMED_NULL for the GET operation.

BINDVERB_POST
: Perform an HTTP POST operation. The data to be posted should be specified in the *stgmedData* member of the BINDINFO structure.

BINDVERB_PUT
: Perform an HTTP PUT operation. The data to put should be specified in the *stgmedData* member of the BINDINFO structure.

BINDVERB_CUSTOM
: Perform a custom operation that is protocol specific. See the *szCustomVerb* member of the BINDINFO structure. The data to be used in the custom operation should be specified in the *stgmedData* member of the BINDINFO structure.

See Also

IBindStatusCallback::GetBindInfo, BINDINFO

BSCF

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

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

Elements

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.

BSCF_INTERMEDIATEDATANOTIFICATION
Identifies an intermediate call to **IBindStatusCallback::OnDataAvailable** for a bind operation.

See Also

IBindStatusCallback::OnDataAvailable

BSCO_OPTION

A client of a binding operation provides values from the **BSCO_OPTION** enumeration when calling the RegisterBindStatusCallback function to register the client's IBindStatusCallback interface. The **BSCO_OPTION** values specify the callback notifications that the client wants from the moniker. The values correspond to methods in the **IBindStatusCallback** interface. Simple clients of **IMoniker::BindToStorage** that want nothing but the data bits need specify only BSCO_ONDATAAVAILABLE.

```
typedef enum tagBSCO_OPTION {
    BSCO_ONSTARTBINDING,
    BSCO_GETPRIORITY,
    BSCO_ONLOWRESOURCE,
    BSCO_ONPROGRESS,
    BSCO_ONSTOPBINDING,
    BSCO_GETBINDINFO,
    BSCO_ONDATAAVAILABLE,
    BSCO_ONOBJECTAVAILABLE,
    BSCO_ALLONIBSC
} BSCO_OPTION;
```

Elements

BSCO_ONSTARTBINDING
The client would like to receive the **OnStartBinding** callback.

BSCO_GETPRIORITY
The client would like to receive the **GetPriority** callback.

BSCO_ONLOWRESOURCE
The client would like to receive the **OnLowResource** callback.

BSCO_ONPROGRESS
The client would like to receive the **OnProgress** callback.

BSCO_ONSTOPBINDING
The client would like to receive the **OnStopBinding** callback.

BSCO_GETBINDINFO
The client would like to receive the **GetBindInfo** callback.

BSCO_ONDATAAVAILABLE
The client would like to receive the **OnDataAvailable** callback.

BSCO_ONOBJECTAVAILABLE
The client would like to receive the **OnObjectAvailable** callback.

BSCO_ALLONIBSC
The client would like to receive all callbacks.

See Also

IBindStatusCallback, RegisterBindStatusCallback

DOCMISC

The DOCMISC enumeration provides miscellaneous status information about a document object. A combination of values from DOCMISC is returned in *pdwStatus* in IOleDocument::GetDocMiscStatus.

```
typedef enum
{
    DOCMISC_CANCREATEMULTIPLEVIEWS    =  1,
    DOCMISC_SUPPORTCOMPLEXRECTANGLES  =  2,
    DOCMISC_CANTOPENEDIT,             =  4,
    DOCMISC_NOFILESUPPORT             =  8
} DOCMISC;
```

Members

DOCMISC_CANCREATEMULTIPLEVIEWS
Object supports multiple views.

DOCMISC_SUPPORTCOMPLEXRECTANGLES
Object supports complex rectangles and, therefore, IOleDocumentView::SetRectComplex.

DOCMISC_CANTOPENEDIT
Object supports activation in a separate windows and, therefore, IOleDocumentView::Open.

DOCMISC_NOFILESUPPORT
Object does not support file read/write.

Remarks

Objects that have a limited user interface for activation purposes should set DOCMISC_CANTOPENEDIT. Those that only support **IPersistStorage** as a persistence mechanism should specify DOCMISC_NOFILESUPPORT. Otherwise, an object must also implement **IPersistFile**.

If an object desires none of these status bits it must return a zero in the **pdwStatus* parameter of **IOleDocument::GetDocMiscStatus**.

See Also

IOleDocument::GetDocMiscStatus, IOleDocumentView::Open, IOleDocumentView::SetRectComplex, **IPersistFile**, **IPersistStorage**

GUIDKIND

The **GUIDKIND** enumeration values are flags used to specify the kind of information requested from an object in the IProvideClassInfo2 and IProvideClassInfo3 interfaces.

```
typedef enum tagGUIDKIND
{
    GUIDKIND_DEFAULT_SOURCE_DISP_IID  =  1,
    GUIDKIND_DEFAULT_SOURCE_IID       =  2,
    GUIDKIND_DEFAULT_DISP_IID         =  3,
    GUIDKIND_DEFAULT_IID              =  4,
    GUIDKIND_TLBID                    =  5,
    GUIDKIND_CLSID                    =  6
} GUIDKIND;
```

Members

GUIDKIND_DEFAULT_SOURCE_DISP_IID
: The interface identifier (IID) of the object's outgoing dispinterface, labeled [source, default]. The outgoing interface in question must be derived from **IDispatch**.

GUIDKIND_DEFAULT_SOURCE_IID
: The interface identifier (IID) of the object's outgoing interface, labeled [source, default]. The outgoing interface can be any COM interface.

GUIDKIND_DEFAULT_DISP_IID
: The interface identifier (IID) of the object's [default] dispinterface that best represents the object as a whole. This dispinterface must be derived from **IDispatch.**.

GUIDKIND_DEFAULT_IID
: The interface identifier (IID) of the object's [default] interface that best represents the object as a whole. This interface can be any COM interface.

GUIDKIND_TLBID
: The GUID identifying the object's current type library.

GUIDKIND_CLSID
: The object's CLSID.

Remarks

Any object implementing **IProvideClassInfo3** should support at least GUIDKIND_DEFAULT_IID through this method.

See Also

IProvideClassInfo2, IProvideClassInfo3

HLBWIF

The **HLBWIF** enumeration contains values relating to the locations and sizes of frame- and document-level windows within a browse context. The HLBWINFO structure is retrieved from the browse context using IHlinkBrowseContext::GetBrowseWindowInfo, and put into the browse context using IHlinkBrowseContext::SetBrowseWindowInfo. Hyperlink targets retrieve the **HLBWINFO** structure during IHlinkTarget::Navigate in order to reposition their user interface properly and ensure as seamless a transition as possible to the new document or object.

```
typedef enum tagHLBWIF
{
    HLBWIF_HASFRAMEWNDINFO,
    HLBWIF_HASDOCWNDINFO,
    HLBWIF_FRAMEWNDMAXIMIZED,
    HLBWIF_DOCWNDMAXIMIZED
} HLBWIF;
```

Members

HLBWIF_HASFRAMEWNDINFO
: This browse context has available frame-level window positioning information.

HLBWIF_HASDOCWNDINFO
: This browse context has available document-level window positioning information.

HLBWIF_FRAMEWNDMAXIMIZED
: Only useful in combination with HLBWIF_HASFRAMEWNDINFO. Indicates that frame-level windows of the browse context should appear maximized.

HLBWIF_DOCWNDMAXIMIZED
: Only useful in combination with HLBWIF_HASDOCWNDINFO. Indicates that document-level windows of the browse context should appear maximized.

See Also

HLBWINFO, IHlinkBrowseContext::GetBrowseWindowInfo, IHlinkBrowseContext::SetBrowseWindowInfo

HLFNAMEF

The **HLFNAMEF** enumeration constants specify which friendly name a client is requesting. These flags are used by the IHlink::GetFriendlyName interface.

```
typedef enum tagHLFNAMEF
{
    HLFNAMEF_DEFAULT,
    HLFNAMEF_TRYCACHE,
    HLFNAMEF_TRYPRETTYTARGET,
    HLFNAMEF_TRYFULLTARGET,
    HLFNAMEF_TRYWIN95SHORTCUT
} HLFNAMEF ;
```

Members

HLFNAMEF_DEFAULT
Requests the cached friendly name, or else the simplified display name.

HLFNAMEF_TRYCACHE
Requests the friendly name that is cached in the Hlink object.

HLFNAMEF_TRYPRETTYTARGET
Requests a beautified version of the display name of the hyperlink target.

HLFNAMEF_TRYFULLTARGET
Requests the full display name of the hyperlink target.

HLFNAMEF_TRYWIN95SHORTCUT
Requests a simplified version of the full display name of the hyperlink target.

See Also

IHlink::GetFriendlyName

HLID

The **HLID** enumeration constants identify the logical positions of a hyperlink identifier in the a hyperlink navigation stack. These constants are used in the IHlinkBrowseContext interface.

```
typedef enum tagHLID
{
    HLID_PREVIOUS,
    HLID_NEXT,
    HLID_CURRENT,
    HLID_STACKBOTTOM,
    HLID_STACKTOP
} HLID;
```

Members

HLID_PREVIOUS
: The hyperlink prior to the current one. If the current hyperlink is the first or only hyperlink in the navigation stack, or if there are no hyperlinks in the navigation stack, there is no previous hyperlink, and methods such as IHlinkBrowseContext::GetHlink will return NULL and E_FAIL when passed this value.

HLID_NEXT
: The hyperlink after the current one. If the current hyperlink is the last or only hyperlink in the navigation stack, or if there are no hyperlinks in the navigation stack, there is no next hyperlink, and methods such as **IHlinkBrowseContext::GetHlink** will return NULL and E_FAIL when passed this value.

HLID_CURRENT
: The current hyperlink. A browsing tool might offer a command to reload the current page, or to re-center the user interface around the beginning portion of the current hyperlink destination, or to restart animation, sound, or other activity by re-navigating to the current hyperlink.

HLID_STACKBOTTOM
: The very first hyperlink in the navigation stack. If there are no hyperlinks in the navigation stack, there is no stack-bottom hyperlink, and methods such as **IHlinkBrowseContext::GetHlink** will return NULL and E_FAIL when passed this value.

HLID_STACKTOP
: The very last hyperlink in the navigation stack. If there are no hyperlinks in the navigation stack, there is no stack-top hyperlink, and methods such as **IHlinkBrowseContext::GetHlink** will return NULL and E_FAIL when passed this value.

Remarks

For convenience and performance, individual hyperlink objects are often identified in a navigation stack like a browse context or a history/favorites list using a ULONG hyperlink identifier or HLID rather than an IHlink interface pointer. This prevents unnecessary passing of interface pointers across process boundaries in common user-interface scenarios, such as building a drop-down menu or scrollable list of the history, or when testing the current location in the navigation stack to enable *Go Back* and *Go Forward*.

See Also

HLQF, IHlinkBrowseContext::GetHlink, IHlinkBrowseContext::QueryHlink, IHlinkBrowseContext::SetCurrentHlink

HLINKGETREF

The **HLINKGETREF** enumeration constants specify whether the client is requesting the absolute, relative or default reference for the hyperlink target. These constants are used in the IHlink::GetMonikerReference and IHlink::GetStringReference methods.

```
typedef enum tagHLINKGETREF
{
    HLINKGETREF_DEFAULT,
    HLINKGETREF_ABSOLUTE,
    HLINKGETREF_RELATIVE
} HLINKGETREF ;
```

Members

HLINKGETREF_DEFAULT
: Used to specify that the client of the hyperlink wishes to retrieve the default reference for hyperlink target. This depends on whether the hyperlink was initialized as a relative or an absolute reference.

HLINKGETREF_ABSOLUTE
: Used to specify that the client of the hyperlink wishes to retrieve the absolute reference for hyperlink target.

HLINKGETREF_RELATIVE
: Used to specify that the client of the hyperlink wishes to retrieve the relative reference for hyperlink target.

See Also

IHlink::GetMonikerReference, IHlink::GetStringReference

HLINKMISC

The **HLINKMISC** enumeration constants specify whether the hyperlink object is a relative or an absolute hyperlink to its target. These constants are used in the IHlink::GetMiscStatus method.

```
typedef enum tagHLINKMISC
{
    HLINKMISC_ABSOLUTE,
    HLINKMISC_RELATIVE
} HLINKMISC ;
```

Members

HLINKMISC_ABSOLUTE
The given hyperlink object contains an absolute reference to the hyperlink target.

HLINKMISC_RELATIVE
The given hyperlink object contains a relative reference to the hyperlink target.

See Also

IHlink::GetMiscStatus

HLINKWHICHMK

The **HLINKWHICHMK** enumeration constants specify whether a moniker being requested is the moniker for the container document or a base moniker specific to a hyperlink site. These constants are used in the IHlinkSite::GetMoniker method.

```
typedef enum tagHLINKWHICHMK
{
    HLINKWHICHMK_CONTAINER,
    HLINKWHICHMK_BASE
} HLINKWHICHMK ;
```

Members

HLINKWHICHMK_CONTAINER

Used to specify that the hyperlink wishes to retrieve the moniker for the hyperlink container corresponding to a particular hyperlink site.

HLINKWHICHMK_BASE

Used to specify that the hyperlink wishes to request the base moniker corresponding to the particular hyperlink site. (These may be different, for example, if a <BASE> tag is used in HTML).

See Also

IHlinkSite::GetMoniker

HLNF

Values from the **HLNF** enumeration are used to indicate how the hyperlink navigation is to proceed, and also convey contextual information about the navigation from each of the objects participating in the navigation protocol to the others. These constants are used in the IHlink, IHlinkBrowseContext, IHlinkFrame, and IHlinkTarget interfaces and in the HlinkNavigateToStringReference, HlinkOnNavigate, HlinkSimpleNavigateToMoniker, and HlinkSimpleNavigateToString API functions.

```
typedef enum tagHLNF
{
    HLNF_INTERNALJUMP             =  1,
    HLNF_NAVIGATINGBACK           =  2,
    HLNF_NAVIGATINGFORWARD        =  4,
    HLNF_USEBROWSECONTEXTCLONE    =  8,
    HLNF_OFFSETWINDOWORG          = 16,
    HLNF_OPENINNEWWINDOW          = 24,
    // (HLNF_USEBROWSECONTEXTCLONE | HLNF_OFFSETWINDOWORG),
    HLNF_CREATENOHISTORY          = 32,
    HLNF_NAVIGATINGTOSTACKITEM    = 64
} HLNF;
```

Members

HLNF_INTERNALJUMP

The navigation is an internal jump within the current hyperlink target. The system provided hyperlink object adds this flag to the *grfHLNF* passed to its IHlink::Navigate prior to calling IHlinkTarget::Navigate when it determines that its relative moniker is NULL. Sending this flag on to the hyperlink target allows the target to exclude any expensive operations and avoid spurious repainting during **IHlinkTarget::Navigate**.

HLNF_NAVIGATINGBACK

The navigation is occurring due to the *Go Back* command, in which case no history should be created in the browse context, and the current position in the navigation stack should be moved back one element. Hyperlink frames and hyperlink containers send this flag to **IHlink::Navigate** for their *Go Back* command.

HLNF_NAVIGATINGFORWARD

The navigation is occurring due to the *Go Forward* command, in which case no history should be created in the browse context, and the current position in the navigation stack should be moved forward one element. Hyperlink frames and hyperlink containers send this flag to **IHlink::Navigate** for their *Go Forward* command.

HLNF_USEBROWSECONTEXTCLONE

When called in **IHlink::Navigate**, the passed in IHlinkBrowseContext should be immediately cloned (via IHlinkBrowseContext::Clone) and used for all subsequent browse context calls and parameters to other methods.

HLNF_OFFSETWINDOWORG

Indicates that the hyperlink target should offset its frame- and/or document-level window(s) from the position returned in the HLBWINFO structure by IHlinkBrowseContext::GetBrowseWindowInfo during **IHlinkTarget::Navigate**. This flag is often passed in conjunction with HLNF_USEBROWSECONTEXTCLONE to implement an Open in New Window command.

HLNF_OPENINNEWWINDOW

An abbreviation for two commonly coincident options: HLNF_USEBROWSECONTEXTCLONE and

HLNF_OFFSETWINDOWORG.

HLNF_CREATENOHISTORY
: Indicates that the browse context should not during IHlinkBrowseContext::OnNavigateHlink add this hyperlink to the navigation stack.

HLNF_NAVIGATINGTOSTACKITEM
: Indicates that the browse context should not during **IHlinkBrowseContext::OnNavigateHlink** add this hyperlink to the navigation stack, and further that it should update its current position to reflect that this hyperlink is the current hyperlink. This flag is used when, for example, the user selects a particular hyperlink from the navigation stack - the user should navigate to the location, but the jump should not be recorded in the navigation stack, and the availability of the *Go Forward* and *Go Back* commands should be reevaluated.

See Also

IHlink::Navigate, IHlinkFrame::Navigate, IHlinkFrame::OnNavigate, IHlinkTarget::Navigate, IHlinkBrowseContext::OnNavigateHlink, HlinkNavigateToStringReference, HlinkOnNavigate, HlinkSimpleNavigateToMoniker, HlinkSimpleNavigateToString

HLQF

The **HLQF** enumeration constants specify query flags used by IHlinkBrowseContext::QueryHlink to determine the state of a particular hyperlink.

```
typedef enum tagHLQF
{
    HLQF_ISVALID    =  1,
    HLQF_ISCURRENT  =  2
} HLQF;
```

Members

HLQF_ISVALID
: Tests the validity of a particular hyperlink. The uHLID parameter may specify either a specific hyperlink within the navigation stack or a relative hyperlink, such as HLID_NEXT or HLID_PREVIOUS.

HLQF_ISCURRENT
: Tests if the specific hyperlink (identified by the uHLID parameter) is the user's current position within the navigation stack.

See Also

HLID, IHlinkBrowseContext::QueryHlink

HLSR

The **HLSR** enumeration constants specify which of the special hyperlink references to choose. These constants are used by the HlinkGetSpecialReference and HlinkSetSpecialReference API functions.

```
typedef enum tagHLSR
{
    HLSR_HOME,
    HLSR_SEARCHPAGE,
    HLSR_HISTORYFOLDER
} HLSR ;
```

Members

HLSR_HOME
Specifies the hyperlink reference to the global user “home” page.

HLSR_SEARCHPAGE
Specifies the hyperlink reference to the global user “search page.”

HLSR_HISTORYFOLDER
Specifies the hyperlink reference to the global user “history folder” page.

See Also

HlinkGetSpecialReference, HlinkSetSpecialReference

INTERNETFLAG

The **INTERNETFLAG** enumeration values are flags used to specify one of the GUID groups in the IProvideClassInfo3::GetFlags method. There is only one bit flag currently defined for the **INTERNETFLAG** group.

```
typedef enum tagINTERNETFLAG
{
    INTERNETFLAG_USESDATAPATHS    =    0x00000001
} INTERNETFLAG;
```

Members

INTERNETFLAG_USESDATAPATHS

Specifies that the object uses one or more data path properties to manage BLOB data storage.

OLECMDF

Designates the type of support provided by an object for the command specified in OLECMD.

```
typedef enum
{
    OLECMDF_SUPPORTED   =  1,
    OLECMDF_ENABLED     =  2,
    OLECMDF_LATCHED     =  4,
    OLECMDF_NINCHED     =  8
} OLECMDF;
```

Members

OLECMDF_SUPPORTED
The command is supported by this object.

OLECMDF_ENABLED
The command is available and enabled.

OLECMDF_LATCHED
The command is an on-off toggle and is currently on.

OLECMDF_NINCHED
TDB

Remarks

Values from the OLECMDF enumeration are used to fill the value of the *cmdf* field in **OLECMD** structures passed to IOleCommandTarget::QueryStatus.

See Also

IOleCommandTarget::QueryStatus, OLECMD, OLECMDF

OLECMDTEXTF

Specifies the type of information that an object should store in the OLECMDTEXT structure passed in IOleCommandTarget::QueryStatus. One value from this enumeration is stored the *cmdtextf* member of the **OLECMDTEXT** structure to indicate the desired information.

```
typedef enum
{
    OLECMDTEXTF_NONE   =  0,
    OLECMDTEXTF_NAME   =  1,
    OLECMDTEXTF_STATUS =  2
} OLECMDTEXTF;
```

Members

OLECMDTEXTF_NONE
No extra information is requested.

OLECMDTEXTF_NAME
The object should return the localized name of the command.

OLECMDTEXTF_STATUS
The object should return a localized status string for the command.

See Also

IOleCommandTarget::QueryStatus, OLECMDTEXT

OLECMDEXECOPT

Specifies command-execution options. One value from this enumeration is passed in the *nCmdExecOpt* argument of IOleCommandTarget::Exec.

```
typedef enum
{
    OLECMDEXECOPT_DODEFAULT      =  0,
    OLECMDEXECOPT_PROMPTUSER     =  1,
    LECMDEXECOPT_DONTPROMPTUSER  =  2,
    OLECMDEXECOPT_SHOWHELP       =  3
} OLECMDEXECOPT;
```

Members

OLECMDEXECOPT_PROMPTUSER
: Execute the command after taking user input.

OLECMDEXECOPT_DONTPROMPTUSER
: Execute the command without prompting the user (for example, clicking on the Print toolbar button, causes the document to be immediately printed without requiring the user input).

OLECMDEXECOPT_DODEFAULT
: Caller is not sure whether the user should be prompted or not.

OLECMDEXECOPT_SHOWHELP
: Object should show help for the corresponding command and not execute

See Also

IOleCommandTarget::Exec

OLECMDID

Specifies which command to execute. (The commands in this enumeration are the standard commands defined by Office 95™.) A single value from this enumeration is passed in the *nCmdID* argument of IOleCommandTarget::Exec.

```
typedef enum
{
     OLECMDID_OPEN            =  1,
     OLECMDID_NEW             =  2,
     OLECMDID_SAVE            =  3,
     OLECMDID_SAVEAS          =  4,
     OLECMDID_SAVECOPYAS      =  5,
     OLECMDID_PRINT           =  6,
     OLECMDID_PRINTPREVIEW    =  7,
     OLECMDID_PAGESETUP       =  8,
     OLECMDID_SPELL           =  9,
     OLECMDID_PROPERTIES      = 10,
     OLECMDID_CUT             = 11,
     OLECMDID_COPY            = 12,
     OLECMDID_PASTE           = 13,
     OLECMDID_PASTESPECIAL    = 14,
     OLECMDID_UNDO            = 15,
     OLECMDID_REDO            = 16,
     OLECMDID_SELECTALL       = 17,
     OLECMDID_CLEARSELECTION  = 18,
     OLECMDID_ZOOM            = 19,
     OLECMDID_GETZOOMRANGE    = 20
} OLECMDID;
```

Members

OLECMDID_OPEN
  File Open

OLECMDID_NEW
  File New

OLECMDID_SAVE
  File Save

OLECMDID_SAVEAS
  File Save As

OLECMDID_SAVECOPYAS
  File Save Copy As

OLECMDID_PRINT
  File Print

OLECMDID_PRINTPREVIEW
  File Print Preview

OLECMDID_PAGESETUP
  File Page Setup

OLECMDID_SPELL
  Tools Spelling

OLECMDID_PROPERTIES

File Properties

OLECMDID_CUT
Edit Cut

OLECMDID_COPY
Edit Copy

OLECMDID_PASTE
Edit Paste

OLECMDID_PASTESPECIAL
Edit Paste Special

OLECMDID_UNDO
Edit Undo

OLECMDID_REDO
Edit Redo

OLECMDID_SELECTALL
Edit Select All

OLECMDID_CLEARSELECTION
Edit Clear

OLECMDID_ZOOM
View Zoom (see below for details)

OLECMDID_GETZOOMRANGE
Retrieves zoom range applicable to View Zoom (see below for details)

Remarks

In OLE Documents technology, an object that is being edited in-place disables the Zoom control on its toolbar and the Zoom command on its View.menu because, logically, the Zoom command applies to the container document , not to the object. Now, in OLE Document Objects technology, the OLECMDID_ZOOM and OLECMDID_GETZOOMRANGE commands provide a document object with a means of notifying the container's frame object of the zoom range that it should display in its user interface.(the container frame is the client-side object that implements **IOleInPlaceFrame** and, optionally, IOleCommandTarget).

The OLECMDID_ZOOM command takes one LONG argument as input and returns one LONG argument on output. This command is used for three purposes:

- *To query the current zoom value* that the caller passes to OLECMDEXECOPT_DONTPROMPTUSER as the execute option in *nCmdExecOpt* and NULL for *pvIn*. The object returns the current zoom value in *pvaOut*. When the object goes UI active, it retrieves the current zoom value from the container's frame object using this same mechanism and updates its zoom control with the returned value.
- *To display the Zoom dialog box* the caller passes OLECMDEXECOPT_PROMPTUSER in *nCmdExecOpt*. The caller can optionally pass the initial value for the dialog box through *pvaIn*, otherwise *pvaIn* must be NULL. If the user presses CANCEL, the object returns OLECMDERR_E_CANCELED; if the user presses OK, then the object returns the user selected value in *pvaOut.* When user selects the View.Zoom menu item, the object calls container's frame object in the same manner. The container then zooms the document to the user selected value, and the object updates its Zoom control with that value.
- *To set a Zoom value* the caller passes OLECMDEXECOPT_DONTPROMPTUSER in *nCmdExecOpt* and passes the zoom value to apply through *pvaIn.* The object validates and normalizes the new value and returns the validated value in *pvaOut.* When the user selects a new zoom value (using the Zoom control on the toolbar for instance), the object calls the container's frame object in this manner. The container zooms the document to the normalized value and the

object updates the Zoom control with that value.

The OLECMDID_GETZOOMRANGE command is used to determine the range of valid zoom values from an object that implements **IOleCommandTarget**. The caller passes MSOCMDEXECOPT_DONTPROMPTUSER in *nCmdExecOpt* and NULL for *pvaIn*. The object returns its zoom range as a DWORD in *pvaOut* where the HIWORD contains the maximum zoom value and the LOWORD contains the minimum zoom value. Typically this command is used when the user drops down the Zoom control on the toolbar of the UI-active object. The applications and objects that support this command are required to support all the integral zoom values that are within the (min,max) pair they return.

See Also

IOleCommandTarget, IOleCommandTarget::Exec, **IOleInPlaceFrame**

PRINTFLAG

Specifies printing options. A combination of values from **PRINTFLAG** is passed in the *grfFlags* parameter of IPrint::Print.

```
typedef enum
{
    PRINTFLAG_MAYBOTHERUSER         =  1,
    PRINTFLAG_PROMPTUSER            =  2,
    PRINTFLAG_USERMAYCHANGEPRINTER  =  4,
    PRINTFLAG_RECOMPOSETODEVICE     =  8,
    PRINTFLAG_DONTACTUALLYPRINT     = 16,
    PRINTFLAG_FORCEPROPERTIES       = 32,
    PRINTFLAG_PRINTTOFILE           = 64
} PRINTFLAG;
```

Members

PRINTFLAG_MAYBOTHERUSER
: User interaction permitted. If this flag is not set, no part of the printing process may interact with the user.

PRINTFLAG_PROMPTUSER
: Prompt the user for job-specific printing options, using the normal print dialog for the object. Support for this option is required. Only valid if PRINTFLAG_MAYBOTHERUSER is specified.

PRINTFLAG_USERMAYCHANGEPRINTER
: Only valid if PRINTFLAG_PROMPTUSER is specified. Indicates that the user may change the target printer; in the absence of this flag, the user must print on the default printer.

PRINTFLAG_RECOMPOSETODEVICE
: The object should attempt to recompose itself to the indicated target device. In the absence of this flag, the object should, if possible, retain any existing compositional-device association that it may have.

PRINTFLAG_DONTACTUALLYPRINT
: Carry out actions resulting from user actions or object-recomposition, but don't actually carry out the printing operation.

PRINTFLAG_PRINTTOFILE
: Print to the file that is named in the *portname* field of DVTARGETDEVICE.

See Also

IPrint::Print

Copyrights and Trademarks

Microsoft Internet Explorer Author's Guide and HTML Reference

This author's guide and HTML reference describes the construction of interactive HTML-based documents that can be viewed with Microsoft® Internet Explorer. The guide begins with a chapter that describes the elements necessary to create the basic HTML document. The chapters that follow include topics for lists, tables, frames, objects, ActiveX Controls, and so on. The HTML Reference section provides a description, an example, and a source specification for each HTML element supported by Internet Explorer.

Category	Description
About HTML Standards	Describes HTML Standards and source specifications.
Basics	Reviews construction of a basic HTML document.
Lists	Reviews the types of lists and their construction.
Character Formatting	Shows character formatting and font control.
Images and Multimedia	Describes how to include graphics, sound, video and marquees.
Color	Gives a list of colors that can be referred to by name.
Frames	Lays out the basics of frame design.
Tables	Describes building and formatting tables and table elements.
Forms	Details the construction of forms and lists the form input controls.
Style Sheets	Illustrates the use of style sheets.
Cascading Style Sheets	Reviews cascading style sheets.
Image Maps	Demonstrates the use of an image with multiple "hot spots".
Scripts	Shows how to include VB and Java scripts.
Objects	Gives an overview of inserting OLE objects and ActiveX Controls.
Client Pull	Describes pulling data from the server automatically.
Character Sets	Appendix A: Special characters.
Source Specification	Appendix B: Source specifications for elements.

About HTML Standards

HTML is the standard language for creating documents for the World Wide Web (WWW). The HTML used by Microsoft Internet Explorer provides a robust implementation of the HTML standards being defined by the main Web organizing bodies and used by the most popular browsers—such as Internet Explorer and Netscape Navigator. HTML for Internet Explorer 3.0 is consistent with published standards, and yet, includes new, more powerful, elements and controls. This HTML recognizes that authors who write on-line documents for the WWW are proactive in promoting additions and modifications to HTML. These authors want the most popular of those extensions to be part of the development language and the browser that they use.

The HTML used by Microsoft Internet Explorer provides key handles for the experienced WWW author to move outside HTML into scripts, such as VBScript and JavaScript, and object-oriented (OO) programming languages, such as Microsoft® Visual C++ and Java. The author may not write these OO programs, but this HTML gives the author the ability to embed objects, control their inputs and influence the layout of these objects within a HTML page. Thus, a spectrum of complexity and power is available to the Web developer with HTML as a foundation. One author can produce a document for the Web using just HTML. Another author who wants to tie a Web page to a database can add VBScript objects. A third author can construct content using a c++ object model. HTML can provide the entry point for each of these authors.

Basics

An HTML document is a text file that contains the elements Internet Explorer uses to display text, multimedia objects, and hyperlinks. HTML stands for hypertext markup language. Using HTML, an author can format a document for display and add hyperlink jumps to other documents. Text which is formatted as a hyperlink can be selected by a user with the mouse. Once selected, the hyperlink jump will load the referenced document into your browser. A hyperlink and the object to which the link jumps can both be defined using HTML.

An element is the most basic part of HTML. An element consists of a start-tag, an end-tag, and the data characters enclosed by the two tags. A tag starts with a less-than (<) sign and ends with a greater-than (>) sign. An end-tag consists of the tag name immediately preceded by a slash (/). Some tags require that you always provide the matching end-tag; others allow you to omit the end tag if the result is clear and unambiguous. For example, here is an element in HTML—a sentence which will dislay in bold:

```
<B>This sentence displays in bold.</B>
```

The example is an HTML element. The start-tag is <B>. The end-tag is </B>. The data characters are "This sentence displays in bold." This element, when read by Internet Explorer will turn on bold formatting, based on the start-tag. The data characters will be displayed in bold. The end-tag switches off the bold formatting. Many elements can be "nested" by placing an entire element inside the tags of another. For example, here is some italicized text placed inside a bold element:

```
<B>This sentence,<I>as written</I>, displays in bold.</B>
```

The <I>as written</I> element displays in both bold and italic because it is nested inside a bold element.

An element can have one or more attributes. An attribute is a parameter associated with an element that extends its meaning. Tags and attribute names are not case-sensitive, but they are typically written in uppercase to distinguish them from the data characters.

This is a very simple HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">lt;HTML>
<HEAD>
<TITLE>Simple HTML Document</TITLE>
</HEAD>
<BODY>
<P>A very simple HTML document.
</BODY>
</HTML>
```

Every HTML document begins with the !DOCTYPE element. !DOCTYPE specifies to the browser which version of HTML is being used to the browser. The next element in this basic document is the HTML element which informs the browser that the content of the file is written in HTML. The matching end-tag (</HTML>) is the last tag in the file. The HEAD tag marks beginning of the document header. The document header describes the elements that apply for all sections of the current document and any documents that contain content or are related to this document. Typically, the TITLE element appears in the header. Internet Explorer displays the text of the TITLE element in its title bar. A menu bar or image that is repeated for other documents may appear in the header section. The BODY element appears at the start of the main content of the document. The BODY element encloses the body text, images, and multimedia objects. The P element inserts a new paragraph with a carriage return and line feed. The end-tag, /P, is typically omitted.

With HTML, you can create hyperlink jumps between your documents. A hyperlink is any text or image which, when clicked, loads another document or another section of the current document into the Internet Explorer window. The A element, or anchor, associates text or a graphic to another document or to a location within the current document. A hyperlink appears as a clickable "hot spot" (the clickable text or image). To create a hyperlink, you enclose the text or image with the anchor tags and set the HREF= attribute to the destination address, as in the following:

```
<P>Click <A HREF="//www.microsoft.com/">here</A> to visit the Microsoft Web
site.
```

In this example, the address for the Web site is enclosed in double quotation marks. The double quotation marks are optional unless the attribute value contains spaces. If you enclose a value that contains double quotation marks, use " for each occurrence of the mark within the value.

For example, you can create a hyperlink destination (anchor spot) within your HTML document by using the NAME= attribute. Use the A element to relate text or a graphic to a name that you create. Then reference the name with a hyperlink. In the following example, the first line creates the named reference. The second line in the example includes a hyperlink with a jump to that place in the document.

```
<A NAME="using"></A><H2>Using Internet Explorer 3.0</H2>

...

<P>For more information, see <A HREF="#usingie30">Using Internet Explorer
3.0</A>
```

When you click the hyperlink "Using Internet Explorer", you jump to the named reference usingie30.

Note Although Internet Explorer can display incomplete or improperly tagged files, the result is often not what you may have intended. You should always use the tags carefully, using them only in the context in which they are defined to be used and omitting end-tags only if they are defined as optional.

The typical HTML document consists of one or more text paragraphs organized into sections. You can mark the beginning of the sections in your HTML documents by using the header Hn element, where "n" is a number from 1 to 6 (1 creates the highest level heading and uses the largest font size). These elements create headings, by applying changes to the size and style of the text to indicate the section level. The specific heading format can be controlled by using attributes or a style sheet; otherwise, it takes the default formatting. The following example creates a first-level section heading:

```
<H1>Welcome to Internet Explorer!</H1>
```

The Hn element allows for six levels; you specify the level by using the element name that includes that level number (H1, H2, H3, and so on). The end-tag is always required.

By default, section headings are left-aligned. You can override the default alignment and center the heading by using the ALIGN= attribute, as in the following example:

```
<H2 ALIGN=CENTER>How to Use Internet Explorer</H2>
```

In addition to using the P element to create simple paragraphs, you can use elements, such as BLOCKQUOTE, LISTING, PLAINTEXT, PRE and XMP, to create paragraphs that use a different size and style for the text. For example, you can use the PRE element to display characters in a fixed-width font rather than the variable-width font used for simple paragraphs, or the BLOCKQUOTE element to slightly indent the paragraph text (from both the left and right margins) to make the paragraph stand out.

You can apply a style to a sequence of paragraphs (for example, those tagged with P) by enclosing the

paragraphs with a ADDRESS, BLOCKQUOTE, or CENTER element. The following example shows how to center a sequence of simple paragraphs:

```
<CENTER>
<P>This paragraph is centered.

<P>And this paragraph is centered too.
</CENTER>
```

An alternate way to center individual paragraphs is to use the ALIGN= attribute with the P element and set the attribute value to CENTER, as in the following example:

```
<P ALIGN=CENTER>This paragraph is centered.
```

will display as:

This paragraph is centered.

Lists

You can create a variety of lists in your document by using the UL, OL, MENU, and DIR elements in conjunction with the LI element. You can also create definition lists which give you a simple two column list for terms and their definitions.

For example, you can create a bulleted list, consisting of individual items preceded by a bullet character, by using UL and LI, as in the following example:

```
<UL>
<LI>Bulleted Lists
<LI>Ordered Lists
<LI>Directory Lists
<LI>Itemized Lists
<LI>Definition Lists
</UL>
```

This displays as:

- Bulleted Lists
- Ordered Lists
- Directory Lists
- Itemized Lists
- Definition Lists

You use the OL and LI elements to create an ordered list. The ordered list consists of individual items that are sequentially numbered or lettered. To set the style of numbering or lettering, you use the TYPE= attribute in OL. Similarly, you use the START= attribute to set the initial number or letter. By default, the style is decimal numbers starting at 1.

```
<OL>
<LI>Step One.
<LI>Step Two.
<LI>Step Three.
</OL>
```

This displays as:

1. Step One.
2. Step Two.
3. Step Three.

The DIR and LI elements create a directory list, consisting of individual items, none containing more than 20 characters, displayed in columns. The MENU and LI elements create an itemized list consisting of individual items.

A definition list is formatted as a two-column list with terms on the left and their definitions on the right. You use the DL, DT and DD elements to create definition lists. The following example shows how to use these elements to create a list:

```
<DL>
<DT>Cat
<DD>A small domesticated mammal.
<DT>Lizard
<DD>A member of the reptile family found in dry areas.
```

```
</DL>
```

This displays as:

Cat
: A small domesticated mammal.

Lizard
: A member of the reptile family found in dry areas.

Character Formatting

You can use a variety of elements to set the size and style of the text characters. For example, you can use the B or STRONG element to make text bold, and the I or EM element to make text italic. Similarly, you can use the S or STRIKE element to strike out text, and the U element to underline text. The following examples set words and phrases within the paragraphs to bold and italic:

```
<P>This <B>word</B> is bold. This <I>word</I> is italic.
```

This displays as:

This **word** is formatted as bold. This *word* is formatted as italic.

```
<P><STRONG>Text formatted as strong.</STRONG> <EM>Emphasized text.</EM>.
```

This displays as:

Text formatted as strong. *Emphasized text.*

You can apply character formatting to a sequence of paragraphs (for example, those tagged with P) by enclosing the paragraphs with a character formatting element. For example, to make all the text in a sequence of paragraphs bold, do the following:

```
<B>
<P>This text is bold.

<P>And this text is bold too.
</B>
```

Although you can use this technique with simple paragraphs, some elements "block" the effect of the character formatting elements. For example, you cannot make all the text in a table bold by enclosing it in a B element.

You can create superscripts and subscripts by using the SUP and SUB elements. These elements reduce the size of the text and align it at the top or bottom of the current line of text.

You can change the size of the text by using elements such as BIG and SMALL, or by using the SIZE= attribute with the FONT element. The following example increases the size of the word "LARGE" and reduces the word "TINY":

```
<P>Use the <BIG>LARGE</BIG> machine for business,
the <SMALL>TINY</SMALL> machine for personal items.
```

If you use the FONT element to change text size, you can specify either a fixed or relative size. A fixed size is a number in the range 1 through 7. A relative size is a positive or negative number, preceded by the plus (+) or minus (-) sign, that indicates a size that is relative to the base font size, as set using the BASEFONT element. The following example shows the effect of using relative sizes:

```
<BASEFONT SIZE=3> This sets the base font size to 3.
<FONT SIZE="+4"> Now the font size is 7.
<FONT SIZE="-1"> Now the font size is 2.
```

You can also use the FACE= attribute with the FONT element to set the name of the font used for text. Some of the most used fonts are "Arial", "Times New Roman", and "Courier New", but you can use the name of any font installed on the computer on which your HTML document is being viewed. The following example sets the "Arial" font for the text in the section heading:

```
<H1><FONT FACE="ARIAL">Welcome to Internet Explorer!</FONT></H1>
```

If the given font is not available, Internet Explorer uses a default font. To increase the chances that a font of your choice is applied to the text, you can specify more than one font in a FONT element. In this case, Internet Explorer checks for each font (in the order given) before using the default font. In the following example, Internet Explorer checks for "Arial", "Lucida Sans", and "Times Roman" before resorting to the default font:

```
<FONT FACE="Arial,Lucida Sans,Times Roman"> This text will be in either
Arial,
Lucida Sans, or Times Roman, depending on which fonts you have installed on
your system.</FONT>
```

You can apply size and font changes to a sequence of paragraphs by enclosing the paragraphs with the FONT element. As with other character formatting elements, some elements do not accept the effect of the FONT, so this technique does not work for all paragraphs. For example, table elements accept changes to the font's name but size changes are not accepted.

Color

You can set colors in your HTML document by using the color attributes of the BODY, FONT, HR, MARQUEE, and TABLE elements. For example, you can set background color for your document by using the BGCOLOR= attribute with the BODY element as in the following example:

```
<BODY BGCOLOR=WHITE>
<P>This page has a white background.
</BODY>
```

You can specify colors in two ways: by using a color name (as in the example above), or by using numbers to denote a red-green-blue color value.

Internet Explorer supports these color names:

AQUA, BLACK, BLUE, FUCHSIA

GRAY, GREEN, LIME, MAROON

NAVY, OLIVE, PURPLE, RED

SILVER, TEAL, WHITE, YELLOW.

Note This feature is not compatible with Netscape v2.0.

A red-green-blue color value consists of three two-digit hexadecimal numbers, with each number specifying the intensity of the corresponding color. For example, the color value #FF0000 is red because the red number is set to its highest value, FF (255 in the decimal, or base 10, system). Green and blue are set to zero. Similarly, #00FF00 is green and #0000FF is blue. The pound sign (#) is optional. In this example, a red horizontal rule is displayed by the following HTML code:

```
<HR COLOR="#00FF0000">
```

Although red-green-blue color values theoretically allow for many thousands of colors, the actual number of colors available for your HTML document depends on the color capabilities of the devices the document will be viewed on. Many personal computers can display only 16 colors due to the type of video display adapter they use. Always choose colors carefully, and whenever possible test your color choices on a variety of computers.

Images and Multimedia

You can embed images, sounds, and even video clips in your HTML document by using the IMG and BGSOUND elements. And you can apply simple animation to text by using the MARQUEE element.

You use the IMG element to insert images into your document. You specify the image source, typically a BMP, GIF or JPEG file, and specify the image attributes, such as the width and height, alignment, and so on. The following example demonstrates how to display the BMP file, TheEarth.bmp.

```
<IMG SRC="TheEarth.bmp" WIDTH=46 HEIGHT=46 ALT="Picture of the Earth">
```

{ewc msdncd, EWGraphic, grpsweeper 0 /a "sweeper.bmp"}

The SRC= attribute specifies the name of the image file. The file will be sized in a square 46 pixels wide by 46 pixels high. For a text-only browser, the text "Picture of the Earth" will display in place of the GIF file. The ALT= attribute specifies the text to be displayed if the user chooses not to view images.

When you place an IMG element in text, Internet Explorer aligns the surrounding text with the bottom of the image. You can align the text with the top or middle of the image by using the ALIGN= attribute to set the alignment to TOP or MIDDLE. In this case, the image keeps its position within the surrounding text.

You can also use the ALIGN= attribute to have the text flow around the image. For example, setting this attribute to LEFT aligns the image with the left margin and wraps all subsequent text around the right side of the image. Similarly, setting it to RIGHT wraps all subsequent text around the left side. When you use the LEFT or RIGHT alignment, typically it is useful to also use the BR element with the CLEAR= attribute to stop wrapping and force all remaining text below the image, as in the following example:

```
<IMG SRC="TheEarth.gif" ALIGN=LEFT> Here's some text to the right of a
picture.
<BR CLEAR=LEFT>Here's some text beneath the picture.
```

You can also use the IMG element to insert video clips, such as .avi (Audio Video Interleave) video files, into your document. You specify the name of the clip by using the DYNSRC= attribute, as in the following example:

```
<IMG DYNSRC="TheEarth.avi" SRC="TheEarth.gif" WIDTH=46 HEIGHT=46
LOOP=INFINITE ALIGN=RIGHT>
```

As in the example, the LOOP= attribute specifies how often the video clip plays. Here the clip plays continuously because LOOP= is set to INFINITE. You can use the CONTROLS and START= attributes to control whether the clip plays when the document loads or when the user clicks on a "placeholder" image specified by the SRC=.

You can make an image a hyperlink hot spot "anchoring" the IMG element to a reference with the A element. By default, Internet Explorer draws a border around the image to mark it as a hot spot. To remove the border, set the BORDER= attribute in the IMG element to zero.

You can add background sounds or music to your document by using the BGSOUND element. You specify the address of a sound file, such as a .WAV, .AU, or MIDI file, and use the LOOP= attribute to set how often the file plays, as in the following example:

```
<BGSOUND SRC="boing.wav" LOOP=5>You will hear a sound played five times in
a row.
```

You can animate a line of text by using the MARQUEE element. The element automatically scrolls the text, to the left or right, whenever a user views your document. To animate the text, you enclose it in the element and set attributes for scroll direction, type and amount, as in the following example:

```
<MARQUEE DIRECTION=RIGHT BEHAVIOR=SCROLL SCROLLAMOUNT=10
SCROLLDELAY=200>This is a scrolling marquee.</MARQUEE>
```

In this example, the text "This is a scrolling marquee." scrolls from the left margin to the right. After it disappears beyond the right margin, it starts again at the left. The text moves 10 pixels after each 200 millisecond delay.

You can align marquees to the left or right, like images, and also set the background color, height, width, and extra spacing around the marquee.

Frames

Frames give you a way to organize and structure the content of your HTML documents by letting you create compound documents that the user can view within the main window of Internet Explorer. To use frames, you create a document that uses the FRAMESET and FRAME elements to divide the main window into rectangular frames (like panes in a window). Then for each frame, you specify an HTML document that contains the content (text and images) to fill the frame. Floating frames enables you to open a browser within a browser. You can insert a floating frame in the same manner in which you can insert an image on an HTML page. You can specify the size of the frame and its border, and you can align it with other text and images on the page. With frames, you can create sophisticated layouts that add and mix sounds, video, animation and colors. Using two frames in a single page, you can display an index in one frame and the content in another. For example, you can split the main window into two equal frames and fill these with different documents by using the following elements:

```
<HTML>
<HEAD>
<TITLE>Two Equal Frames</TITLE>
</HEAD>
<FRAMESET COLS="50%,*">
<FRAME SRC=x.htm>
<FRAME SRC=y.htm>
</FRAMESET>
</HTML>
```

In this example, the COLS= attribute in the FRAMESET element specifies the width of the frames. The width of the first frame is 50% of the main window, and the width of the second, given as an asterisk, is relative to the first (meaning it spans whatever is left of the main window). Note that this document does not contain a BODY element. This is because documents that define frames do not contain content. Instead, the SRC= attribute in each FRAME element specifies a document. In this example, the X.HTM and Y.HTM are content sources for the frames.

You can divide the main window into rows, as well as columns, by using the ROWS= attribute. Furthermore, you can independently divide individual rows into rows and columns by nesting FRAMESET elements. The following example shows how to divide the main window into two rows in which the last row is divided into two columns:

```
<HTML>
<HEAD>
<TITLE>Nested Frames</TITLE>
</HEAD>
<FRAMESET ROWS="10%,*">
<FRAME SCROLLING=NO SRC=z.htm>
<FRAMESET COLS="50%,*">
<FRAME SRC=x.htm>
<FRAME SRC=y.htm>
</FRAMESET>
</FRAMESET>
</HTML>
```

In this example, the SCROLLING= attribute is used in the first FRAME element to prevent the scroll bar from being displayed. By default, Internet Explorer displays the scroll bar only if the entire content of the frame does not fit within the frame. Setting SCROLLING= to NO always prevents the scroll bar.

The FRAME element has attributes to let you set the width and height of margins within the frame and whether the frame has a border. The FRAMESET element has attributes to let you set the spacing between frames and whether the frames in the set have borders.

An important feature of the FRAME element is the NAME= attribute. This attribute lets you assign a unique name to the frame, and you can use this name when creating hyperlinks to direct documents into the frame. To create such a hyperlink, you need to use the TARGET= attribute in the A element. For example, the following element creates a hyperlink that displays the X.htm file in a frame named CONTENT:

```
<A HREF="X.htm" TARGET="CONTENT">List of Components</A>
```

Internet Explorer provides an alternate way to create compound documents by letting you place FRAME elements in your HTML document. This lets you insert HTML documents into your document in the same way you insert images using the IMG element. This means you can use the ALIGN= attribute to just as you do with IMG to align the frame with the surrounding text. The following example aligns a frame at the left margin and wraps subsequent text around the right side of the frame:

```
<FRAME SRC="x.htm" ALIGN=LEFT> Here's some text to the right of a frame.
<BR CLEAR=LEFT>Here's some text beneath the frame.
```

Tables

Use the TABLE element to format a table. The TR element (table row) inserts a row in the table, and TD element (table detail) inserts a cell within a row. With Internet Explorer 3.0 as the browser, images or text can be placed within the cells. This example shows the HTML elements for a simple table:

```
<TABLE>
<TR>
<TD>Apples<TD>Celery
<TR>
<TD>Oranges<TD>Carrots
</TABLE>
```

This displays as:

Apples	Celery
Oranges	Carrots

By default, Internet Explorer aligns the table to the left. The width of the table, unless specified, is determined by the contents of the longest element in each column. The content of each cell is aligned to the middle and to the left edge of each cell. You can override these defaults. For example, you can set the width of the table by using the WIDTH= attribute in the TABLE element. You can align the content of each cell to the top, left, right, or bottom of a cell by using the ALIGN= and VALIGN= attributes in TR or TD. This example creates a table that is the full width of the Internet Explorer window. The contents of the cells are at the top and the left:

```
<TABLE WIDTH="100%">
<TR VALIGN=TOP ALIGN=LEFT>
<TD>Apples<TD>Celery
<TR VALIGN=TOP ALIGN=LEFT>
<TD>Oranges<TD>Carrots
</TABLE>
```

This displays as:

Apples	Celery
Oranges	Carrots

In this example, the table width is given as a percentage of the total Internet Explorer window width. But you can also specify table widths in pixels.

You can add a caption, row and column headings, and a border to a table by using elements and attributes. For example, you can add a caption to a table by using the CAPTION element. By default, the caption is centered above the table, but you can use the ALIGN= attribute to place the caption at the top or bottom and at the left or right edge of the table.

To add headings to the rows and columns of a table, use the TH element. This element is like the TD element in that it creates a cell and can contain text and images, but it automatically emphasizes its text to distinguish it from text in other cells. To draw a border around the table and the individual cells, you use the BORDER= attribute in the TABLE element. Specify the border width in pixels. The following example creates a table with headings, border, and a caption:

```
<TABLE BORDER=1>
<CAPTION>Fruits and Vegetables</CAPTION>
<TR>
<TH>Fruits<TH>Vegetables
```

```
<TR>
<TD>Apples<TD>Celery
<TR>
<TD>Oranges<TD>Carrots
</TABLE>
```

By default, Internet Explorer centers headings in the cell, but you can override this by using the ALIGN= and VALIGN= attributes.

You can add color to your tables by using the BGCOLOR= and BORDERCOLOR= attributes. These attributes are available in the TABLE, TR, and TD elements, so you can apply colors to all cells in a table, to cells in selected rows, or to individual cells. The BGCOLOR= attribute sets the color used to fill the background of the cell before text and images are drawn. The BORDERCOLOR= attribute sets the color of the borders drawn around the table, row, or cell. The following example uses the same background color for the column headings, but different colors for the two columns in the table:

```
<TABLE BORDERCOLOR=NAVY BORDER=1>
<CAPTION>Fruits and Vegetables</CAPTION>
<TR BGCOLOR=GRAY>
<TH>Fruits<TH>Vegetables
<TR>
<TD BGCOLOR=LIME>Apples<TD BGCOLOR=AQUA>Celery
<TR>
<TD BGCOLOR=LIME>Oranges<TD BGCOLOR=AQUA>Carrots
</TABLE>
```

You can change the character formatting for the text in a table by using elements such as B, I, and FONT. You can change the color and font name for all text in a table by enclosing the table in an appropriate FONT element, but the table elements block the effect of other character formatting elements. To get these effects, you must apply the elements within each cell.

Within a cell, you can use most of the elements that you ordinarily use in the body of the HTML document, including elements for section headings, lists, and even other tables. Using tables in this way can give you additional control over the placement of text and images when your document is displayed, but can also make the management of your document more complex. For example, you can use tables to give your document a two-column layout by nesting a single-column table in each cell of a two-column table. But if you do this, you must take special care to divide the content of your document equally between the two nested tables and be prepared to account for differences in the size of the window through which users view your document. In most cases, documents that use tables in this way are designed to be viewed within a minimum window size at a given screen resolution.

If you use tables in the more traditional way (that is, presenting information in rows and columns), there are some additional elements and attributes that can make that job easier. The THEAD, TBODY, and TFOOT elements let you divide your tables into parts: header, body, and footer. The COLGROUP and COL elements let you group columns within the table and globally apply properties, such as alignment, to the columns without having to specify these properties in each TD element.

The FRAME= and RULES= attributes in the TABLE element let you control how the table border is drawn. For example, you can choose to have no border around the outside of the table while restricting the border inside the table to just vertical rules separating the columns and horizontal rules separating the table header, body, and footer. The COLSPAN= and ROWSPAN= attributes in the TD and TH elements let you extend the content of a cell into adjoining cells. This is useful, for example, if you need to stretch a column heading across more than one column.

The following example shows how some of these elements and attributes can be used in a table:

```
<TABLE WIDTH="50%" BORDER=1 FRAME=BOX RULES=GROUPS>
<COLGROUP ALIGN=CENTER>
<COLGROUP ALIGN=CENTER>
<THEAD>
<TR>
<TH COLSPAN=2>Fruits and Vegetables
<TBODY>
<TR>
<TD>Apples<TD>Celery
<TR>
<TD>Oranges<TD>Carrots
</TABLE>
```

Internet Explorer supports advanced table functionality including displaying background images behind table cells, specifying rules or borders just along columns or along rows, and aligning text to the baseline within a table cell.

Style Sheets

Style sheets give you the ability to attach styles to HTML pages, letting you control margins, line spacing, the placement of elements, colors, fonts faces, and font sizes. Style sheets make it easier to create an index because indexing software has only to read the structural elements rather than the full content of a page. Cascading style sheets are supported by Microsoft Internet Explorer 3.0.

You can include a style sheet as part of an HTML document and apply the style to some or all of the text, or you can create a style sheet as a separate document and attach it to one more pages on your Web site. And you can use both methods in a single document—creating a style sheet for all the documents on a Web site, while selectively applying a special style sheet to text within selected documents.

There are two ways to place style information inside a document. The first is to assign a style to an element. For example, here's how to specify a paragraph with a font size of 20 points.

```
<P STYLE="font-size: 20pt"> This paragraph is in 20-point text.
As Hemingway once said, it is a great thing to be able to specify point
sizes, especially large ones.
```

The second way to place style information in-line is to use a new element called SPAN. SPAN by itself doesn't mean anything; you use it to surround text to which you want to add style information. Here's how the SPAN element might be used:

```
<SPAN STYLE="margin-left: 1.0in"> This paragraph is 1.0 inches from the
left margin.<SPAN>
```

To place style information at the top of a page, insert a STYLE block at the top of your document. The block is placed after HTML element and before the BODY element. For example:

```
<HTML>
<STYLE>
BODY {background: white; color: black}
H1 {font: 14pt Arial bold}
P {font: 10pt Arial; text-indent: 0.5in}
A {text-decoration: none; color: blue}
</STYLE>
<BODY>
<H1>This is a headline! In 14-point Arial bold!</H1>
</BODY>
</HTML>
```

To assign more than one kind of style information at the same time, separate the styles with semicolons. For example, to set an entire HTML page to 10-point Times font, the colors to black on white, and both left and right margins to one inch, place this text before the BODY element:

```
<STYLE>
BODY {font: 10pt Times; color: black; background: white; margin-left: 1in;
margin-right: 1in}
</STYLE>
```

An external style sheet consists of the STYLE element and its attributes. To link a page to this style sheet, use the LINK element, as in the following example (where Mystles.css is the external style sheet):

```
<LINK REL=STYLE TYPE="text/css"
SRC="http://www.mycompany.com/mystyles.css">
```

Note You will need to make sure that the MIME type reported for "mystyles.css" is text/css.

The text formatting features supported by Internet Explorer 3.0 are described in this reference guide under the STYLE element.

Cascading Style Sheets

Using cascading style sheets, more than one style sheet can influence the presentation simultaneously. There are two main reasons for this feature: modularity and author/reader balance.

A style sheet designer can combine several (partial) style sheets to reduce redundancy:

```
@import url(http://www.style.org/stylea);
@import url(http://www.style.org/styleb);

H1 { color: red }     /* override imported sheets */
```

Also, both readers and authors can influence the presentation through style sheets. To do so, they use the same style sheet language thus reflecting a fundamental feature of the web: everyone can become a publisher.

Sometimes conflicts will arise between the style sheets that influence the presentation. Conflict resolution is based on each style rule having a weight. By default, the weights of the reader's rules are less than the weights of rules in the author's documents, but style sheet designers can increase the weights of their rules by using the keyword "important". In this example, the H1 and P styles will override all other style sheet formatting:

```
H1 { color: red ! important }
P  { font-size: 12pt ! important }
```

An reader rule labelled important will override an author rule with normal weight. An author rule labelled important will override an important reader rule. *Cascading order*

Conflicting rules are intrinsic to the CSS mechanism. To find the value for an element/property combination, the following algorithm should be followed:

1. Find all declarations that apply to the element/property in question. Declarations apply if the selector matches the element in question. If no declarations apply, the inherited value is used. If there is no inherited value (this is the case on the root element and for properties that do not inherit), the initial value is used.

2. Sort the declarations by explicit weight: declarations marked '!important' carry more weight than unmarked (normal) declarations.

3. Sort by origin: the author's style sheets override the reader's style sheet which override the UA's default values.

4. Sort by specificity of selector: more specific selectors will override more general ones. To find the specificity, cout the number of ID attributes in the selector (a), the number of CLASS attributes in the selector (b), and the number of tag names in the selector (c). Concatenating the three numbers (in a number system with a large base) gives the specificity. Some examples are:

```
LI            {...}  /* a=0 b=0 c=1 -> specificity =   1 */
UL LI         {...}  /* a=0 b=0 c=2 -> specificity =   2 */
UL OL LI      {...}  /* a=0 b=0 c=3 -> specificity =   3 */
LI.red        {...}  /* a=0 b=1 c=1 -> specificity =  11 */
UL OL LI.red  {...}  /* a=0 b=1 c=3 -> specificity =  13 */
#x34y         {...}  /* a=1 b=0 c=0 -> specificity = 100 */
```

5. Sort by order specified: if two rules have the same weight, the latter specified should live.

The search for the property value can be terminated whenever one rule has a higher weight than the other rules that apply to the same element/property combination.

This strategy gives author's style sheets considerably higher weight than those of the reader. It is therefore important that the reader has the ability to turn off the influence of a certain style sheet, e.g. through a pull-down menu.

A 'STYLE' attribute on an element (see section 1.1 for an example) should be considered as if an ID attribute had been specified at the end of the style sheet.

The UA may choose to honor other stylistic attributes (e.g. 'ALIGN') as if a 'STYLE' attribute had been used. When in conflict with other stylistic attributes, the 'STYLE' attribute should win.

Image Maps

Image maps allow users to access different documents by clicking different areas in an image. You can implement image maps in two ways: by storing image map information on a server or by including image map information in your document.

If you store image map information on a server, you need a script or other service on the server to process click information. In your document, you mark the image as a "server-side" image map by using the ISMAP attribute in the IMG element and enclosing the image in an A element, as in the following example:

```
<A HREF="Jump.map"><IMG SRC="Sample.gif" ISMAP></A>
```

In this example, the image map information is in the file name Jump.map. When the user clicks on the picture in Sample.gif, the server receives the coordinates of the click, and can pick the appropriate destination for the click by checking the information in Jump.map.

If you include image map information in your document, Internet Explorer processes the click information and picks the appropriate destination for the click. In your document, you mark the image as a "client-side" image map by using the USEMAP= attribute in the IMG element, and you add image map information by using the MAP and AREA elements, as in the following example:

```
<MAP NAME="map1">
<AREA SHAPE="RECT" COORDS="0, 0, 16, 16" HREF="Sample1.htm">
<AREA SHAPE="RECT" COORDS="16, 0, 16, 16" NOHREF>
<AREA SHAPE="RECT" COORDS="0, 16, 16, 16" HREF="Sample2.htm">
<AREA SHAPE="RECT" COORDS="16, 16, 16, 16" HREF="Sample3.htm">
</MAP>
<IMG BORDER=0 SRC="map1.gif" USEMAP="#map1">
```

In this example, the image map defines four equal rectangular areas. One area has no corresponding destination, but the other three map to the Sample1.htm, Sample2.htm, and Sample3.htm files, respectively.

The AREA element permits other shapes, such as circles and polygons. If two or more shapes overlap, Internet Explorer uses the first shape defined in the MAP element to determine the destination. Any number of AREA elements can specify the same destination. This is useful if you want to map a complex shape to a single destination. If a portion of the image is not within a given shape, clicking in that portion has no effect.

Forms

Forms provide a way to prompt the user for information and to carry out actions based on that input. A form consists of one or more input controls that the user uses to enter text and makes choices. Once the user provides the input, the form collects the information and sends it to a specified destination on a server. To carry out the requested action, the server must have a script or other service that corresponds to the given destination. This script processes the information and carries out any actions.

To create a form, you use the FORM element to enclose one or more INPUT elements. The FORM element specifies the action to take when the user has provided the information. The INPUT elements define the type and function of the input controls in the form. The following example shows how to combine these elements to create a form:

```
<FORM ACTION="http://intranet/survey" METHOD=POST>
<P>Name
<BR><INPUT NAME="CONTROL1" TYPE=TEXTBOX VALUE="Your Name">
<P>Password
<BR><INPUT TYPE="PASSWORD" NAME="CONTROL2">
<P>Color
<BR><INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="0" CHECKED>Red
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="1">Green
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="2">Blue
<P>Comments
<BR><INPUT TYPE="TEXTAREA" NAME="CONTROL4" SIZE="20,5" MAXLENGTH="250">
<P><INPUT NAME="CONTROL5" TYPE=CHECKBOX CHECKED>Send receipt
<P><INPUT TYPE="SUBMIT" VALUE="OK"><INPUT TYPE="RESET" VALUE="Reset">
</FORM>
```

Scripts

There are three ways to attach and invoke scripts in HTML: contain them in the <SCRIPT> element, use attributes of HTML elements, or use a custom URL type.

Using the SCRIPT Element

Use the SCRIPT element to add scripts to HTML. SCRIPT is a character-like element for embedding script code anywhere in the document HEAD or BODY. The SCRIPT element can be used to reference external scripts, using the SRC attribute, and to include script statements within the HTML document.

HTML documents can include multiple SCRIPT elements that can be placed in the document heading or body. This allows script statements for a form to be placed near the corresponding FORM element.

Here is a simple example of a page that uses the SCRIPT element:

```
<SCRIPT language="VBScript">
    '... Additional VBScript statements ...
</SCRIPT>
```

The same example in JavaScript would read:

```
<SCRIPT language="JavaScript">
    //... Additional JavaScript statements ...
</SCRIPT>
```

Evaluation of SCRIPT

The SCRIPT element is evaluated when the document is loaded, and all code is executed at load time. For functions such as document.write, the order of script elements can affect the output of the document. For example, the page:

```
<HTML><BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write ("Hello world.")
</SCRIPT>
This is a document.
</BODY></HTML>
```

results in:

```
Hello world. This is a document.
```

While the page:

```
<HTML><BODY>
This is a document.
<SCRIPT LANGUAGE="VBScript">
document.write ("Hello world.")
</SCRIPT>
</BODY></HTML>
```

results in:

```
This is a document. Hello world.
```

Also, because script statements are evaluated when the document is loaded, attempts to reference objects will fail if these objects are defined by HTML elements that occur later in the document.

The document object's write method can insert text and objects—such as buttons (defined using the INPUT element) and ActiveX controls (defined using the OBJECT element). These objects can be referenced in any script block following the script block that output them.

Using Scripts as Attributes of HTML Elements

Another way to insert scripts is to add attributes to element tags in HTML. These attributes match with events on the elements, and the script is executed when the event is fired. This method can be used for any HTML intrinsic elements, such as forms, buttons, or links; however, this method does not work for items inserted using the OBJECT element.

The following example uses this syntax in Button1 to handle the onClick event. To demonstrate the ability to combine multiple scripting languages on the same page, the script for Button1 is implemented in VBScript and that for Button2 in JavaScript.

```
<form name="Form1">
  <input type="button" name="Button1" value="   Press me   "
  onClick="pressed" language="VBScript">
  <input type="button" name="Button2" value="Press me too!"
  onClick="pressed2()" language="JavaScript">
</form>

<script language="VBSCRIPT">
  sub pressed
    document.Form1.Button1.value="I'm VBS"
    alert "I've been pressed!"
  end sub
</script>
<script language="JavaScript">
  function pressed2()
  {
    document.Form1.Button2.value="I'm JavaScript"
    alert("Are you impressed?")
  }
</script>
```

Note the use of the language attribute on the input element to indicate the script's language. If no language is specified, the script defaults to the language of the most recently encountered script block. If no script block has been encountered, the language defaults to JavaScript.

The elements FORM, INPUT, BODY, and A support this syntax, but with differing events. See the individual elements referenced later in this document.

An Alternative Form of Using SCRIPT

The SCRIPT element can also be used with the FOR="object" EVENT="eventname" syntax. This method can be used for any named elements, and for any elements inserted using the OBJECT element. The following example is similar to the previous script example, but it uses a different syntax:

```
<form name="Form1">
```

```
  <input type="button" name="Button1" value="Press me">
  <script for="Button1" event="onClick" language="VBScript">
    alert "I've been pressed"
    document.Form1.Button1.value="OUCH"
  </script>
</form>
```

Using Scripts in URLs

Scripts can be invoked using the A element combined with a custom URL type. This allows a script to be executed when the user clicks on a hyperlink. This URL type is valid in any context, but is most useful when used with the A element. For example:

```
<A HREF="javascript:alert('hi there')">Click me to see a message.<A>
```

displays an alert message box that contains the text 'hi there'.

Syntax

script-engine:script-code

Executes the script code using the script engine when the URL is resolved. For example, to execute a script when the user clicks on a hyperlink, use:

```
<title> JavaScript example </title>
<A HREF=" javascript:alert(document.title)">Click here to see the title of
the current document..</A>
```

Notice that the script is executed in the context of the current page, which means that document.title evaluates to the document containing the script.

Argument	Type	Description
script-engine	String	A string that names a scripting engine (*must* be JavaScript for Beta 1).
script-code	String	A string that evaluates to a script in the syntax supported by the scripting engine. This script is executed by the scripting engine when the URL is evaluated.

Note This syntax is only supported for JavaScript in the current build of Internet Explorer; in particular, VBScript: will not work in the current build. All scripting engines will be supported in future builds. Also, the JavaScript: syntax is currently supported only from scripts, not when typed into the address bar by users.

Objects

Internet Explorer supports objects according to the HTML 3.2 object model. Objects add functionality to your HTML document by letting you insert images, video, and programs, such as JAVA applets, and ActiveX controls.

To insert an object, you use the OBJECT element, supplying attribute values that specify the object type, location, initial data, and so on. If the object has configurable properties, you can set these using the PARAM element. The following example shows how to insert the marquee ActiveX control and fill it with content:

```
<OBJECT
ALIGN=CENTER CLASSID="clsid:1a4da620-6217-11cf-be62-0080c72edd2d"
WIDTH=200 HEIGHT=200 BORDER=1 HSPACE=5
ID=marquee>
<PARAM NAME="ScrollStyleX" VALUE="Circular">
<PARAM NAME="ScrollStyleY" VALUE="Circular">
<PARAM NAME="szURL" VALUE="marqcont.htm">
<PARAM NAME="ScrollDelay" VALUE=60>
<PARAM NAME="LoopsX" VALUE=-1>
<PARAM NAME="LoopsY" VALUE=-1>
<PARAM NAME="ScrollPixelsX" VALUE=0>
<PARAM NAME="ScrollPixelsY" VALUE=-3>
<PARAM NAME="DrawImmediately" VALUE=0>
<PARAM NAME="Whitespace" VALUE=0>
<PARAM NAME="PageFlippingOn" VALUE=0>
<PARAM NAME="Zoom" VALUE=100>
<PARAM NAME="WidthOfPage" VALUE=400>
</OBJECT>
```

In this example, the OBJECT element specifies the class identifier of the control (assumed to be already installed and registered) and the alignment, width, height, and other attributes of the control. The series of PARAM elements sets the values for the individual properties of the control, determining how and when the contents are scrolled. The szURL property determines which HTML document is used as content.

A matching end-tag is required for each OBJECT element. Within these elements, you may place one or more PARAM elements. You can also place any elements and text that you would ordinarily use in the body of the HTML document, but these elements and text are not processed and displayed unless the HTML viewer does not process the OBJECT element.

For detailed information about the HTML 3.0 object model, see the document at *http://www.w3.org/pub/WWW/TR/WD-object.html*.

Embed

Microsoft Internet Explorer 3.0 will support the EMBED element syntax for embedding objects on an HTML page. This support is meant for compatibility with other browsers. The accepted WWW Consortium HTML standard recommends using the OBJECT element for embedding objects in HTML, and Internet Explorer 3.0 supports this syntax as well.

Using OBJECT but degrading gracefully to work in other browsers:

Internet Explorer 3.0 supports the OBJECT element. Browsers that comply with the accepted World Wide Web Consortium (W3C) HTML standard will understand this HTML syntax. However, the OBJECT syntax degrades gracefully in other browsers as well, because browsers that are compatible

with the OBJECT element will ignore additional elements placed within the OBJECT elment. The following illustrates how use the OBJECT element but allow content to be viewable by other browsers:

```
OBJECT DATA=&amp;quot;MyMovie.AVI&amp;quot; WITDTH=100 HEIGHT=250&amp;gt;
     <PARAM NAME=AUTOSTART VALUE=TRUE&amp;gt;
     <PARAM NAME=PLAYBACK VALUE=FALSE&amp;gt;
     <EMBED SRC=&amp;quot;MyMovie.AVI&amp;quot; WIDTH=100 HEIGHT=250 AUTOSTART=TRUE
  PLAYBACK=FALSE&amp;gt;
</OBJECT&amp;gt;
```

In browsers that support the OBJECT element, the EMBED element will be ignored. Browsers that do not support OBJECT will ignore the OBJECT element and PARAM attribute.

Client Pull

Client pull provides the ability to automatically load a new document in the specified time or reload a document on a regular basis. Internet Explorer supports client pull using the META element. The META element must be inside the HEAD element of the HTML document. For instance, META HTTP-EQUIV=REFRESH CONTENT=2 reloads a document every two seconds. Setting the HTTP-EQUIV attribute to REFRESH gives the instruction to reload. CONTENT specifices the time in seconds that the page refreshes. You can specify any URL in the element. If no URL is specified, the current document is reloaded. The META element also has several other functions, such as specifying keywords for Web search engines to use for indexxing. Read more about META in the HTML Reference.

Character Sets

This following information is described here:

- ISO Latin-1 Character Set
- Complete Character Set including Microsoft Internet Explorer additions.

ISO Latin-1 Character Set

Character	Decimal Code	HTML	Description
À	&#192;	&Agrave;	Capital A, grave accent
à	&#224;	&agrave;	Small a, grave accent
Á	&#193;	&Aacute;	Capital A, acute accent
á	&#225;	&aacute;	Small a, acute accent
Â	&#194;	&Acirc;	Capital A, circumflex
â	&#226;	&acirc;	Small a, circumflex
Ã	&#195;	&Atilde;	Capital A, tilde
ã	&#227;	&atilde;	Small a, tilde
Ä	&#196;	&Auml;	Capital A, diæresis / umlaut
ä	&#228;	&auml;	Small a, diæresis / umlaut
Å	&#197;	&Aring;	Capital A, ring
å	&#229;	&aring;	Small a, ring
Æ	&#198;	&AElig;	Capital AE ligature
æ	&#230;	&aelig;	Small ae ligature
Ç	&#199;	&Ccedil;	Capital C, cedilla
ç	&#231;	&ccedil;	Small c, cedilla
È	&#200;	&Egrave;	Capital E, grave accent
è	&#232;	&egrave;	Small e, grave accent
É	&#201;	&Eacute;	Capital E, acute accent
é	&#233;	&eacute;	Small e, acute accent
Ê	&#202;	&Ecirc;	Capital E, circumflex
ê	&#234;	&ecirc;	Small e, circumflex
Ë	&#203;	&Euml;	Capital E, diæresis / umlaut
ë	&#235;	&euml;	Small e, diæresis / umlaut
Ì	&#204;	&Igrave;	Capital I, grave accent
ì	&#236;	&igrave;	Small i, grave accent
Í	&#205;	&Iacute;	Capital I, acute accent
í	&#237;	&iacute;	Small i, acute accent
Î	&#206;	&Icirc;	Capital I, circumflex
î	&#238;	&icirc;	Small i, circumflex
Ï	&#207;	&Iuml;	Capital I, diæresis / umlaut
ï	&#239;	&iuml;	Small i, diæresis / umlaut
Ð	&#208;	&ETH;	Capital Eth, Icelandic

ð	ð	ð	Small eth, Icelandic
Ñ	Ñ	Ñ	Capital N, tilde
ñ	ñ	ñ	Small n, tilde
Ò	Ò	Ò	Capital O, grave accent
ò	ò	ò	Small o, grave accent
Ó	Ó	Ó	Capital O, acute accent
ó	ó	ó	Small o, acute accent
Ô	Ô	Ô	Capital O, circumflex
ô	ô	ô	Small o, circumflex
Õ	Õ	Õ	Capital O, tilde
õ	õ	õ	Small o, tilde
Ö	Ö	Ö	Capital O, diæresis / umlaut
ö	ö	ö	Small o, diæresis / umlaut
Ø	Ø	Ø	Capital O, slash
ø	ø	ø	Small o, slash
Ù	Ù	Ù	Capital U, grave accent
ù	ù	ù	Small u, grave accent
Ú	Ú	Ú	Capital U, acute accent
ú	ú	ú	Small u, acute accent
Û	Û	Û	Capital U, circumflex
û	û	û	Small u, circumflex
Ü	Ü	Ü	Capital U, diæresis / umlaut
ü	ü	ü	Small u, diæresis / umlaut
Ý	Ý	Ý	Capital Y, acute accent
ý	ý	ý	Small y, acute accent
Þ	Þ	Þ	Capital Thorn, Icelandic
þ	þ	þ	Small thorn, Icelandic
ß	ß	ß	Small sharp s, German sz
ÿ	ÿ	ÿ	Small y, diæresis / umlaut

Character Set

This table describes the complete character set for Internet Explorer 3.0 English (US.) The first column shows the character as it appears in IE 3.0. The second column shows the decimal number, as it is written in an HTML document, to produce the characters. Occasionally, special characters have nemonic names. For example, the registered trademark character can be written in HTML as ®. The third column lists these HTML characters. The last column gives a description of each character where appropriate.

Character	**Decimal Code**	**HTML**	**Description**
	&#00;		Unused
	&#01;		Unused
	&#02;		Unused
	&#03;		Unused
	&#04;		Unused

	&#05;		Unused
	&#06;		Unused
	&#07;		Unused
	&#08;		Unused
			Horizontal tab
	Line feed
	&#11;		Unused
			Unused
	Carriage Return
	&#14;		Unused
	&#15;		Unused
	&#16;		Unused
	&#17;		Unused
	&#18;		Unused
	&#19;		Unused
	&#20;		Unused
	&#21;		Unused
	&#22;		Unused
	&#23;		Unused
	&#24;		Unused
	&#25;		Unused
	&#26;		Unused
	&#27;		Unused
	&#28;		Unused
	&#29;		Unused
	&#30;		Unused
	&#31;		Unused
			Space
!	!		Exclamation mark
"	"	"	Quotation mark
#	#		Number sign
$	$		Dollar sign
%	%		Percent sign
&	&	&	Ampersand
'	'		Apostrophe
(	(		Left parenthesis
)	)		Right parenthesis
*	*		Asterisk
+	+		Plus sign
,	,		Comma
-	-		Hyphen
.	.		Period (fullstop)
/	/		Solidus (slash)

0	&#48;		Digit 0
1	&#49;		Digit 1
2	&#50;		Digit 2
3	&#51;		Digit 3
4	&#52;		Digit 4
5	&#53;		Digit 5
6	&#54;		Digit 6
7	&#55;		Digit 7
8	&#56;		Digit 8
9	&#57;		Digit 9
:	&#58;		Colon
;	&#59;		Semi-colon
<	&#60;	&lt;	Less than
=	&#61;		Equals sign
>	&#62;	&gt;	Greater than
?	&#63;		Question mark
@	&#64;		Commercial at
A	&#65;		Capital A
B	&#66;		Capital B
C	&#67;		Capital C
D	&#68;		Capital D
E	&#69;		Capital E
F	&#70;		Capital F
G	&#71;		Capital G
H	&#72;		Capital H
I	&#73;		Capital I
J	&#74;		Capital J
K	&#75;		Capital K
L	&#76;		Capital L
M	&#77;		Capital M
N	&#78;		Capital N
O	&#79;		Capital O
P	&#80;		Capital P
Q	&#81;		Capital Q
R	&#82;		Capital R
S	&#83;		Capital S
T	&#84;		Capital T
U	&#85;		Capital U
V	&#86;		Capital V
W	&#87;		Capital W
X	&#88;		Capital X
Y	&#89;		Capital Y

Z	&#90;		Capital Z
[	&#91;		Left square bracket
\	&#92;		Reverse solidus (backslash)
]	&#93;		Right square bracket
^	&#94;		Caret
_	&#95;		Horizontal bar (underscore)
`	&#96;		Acute accent
a	&#97;		Small a
b	&#98;		Small b
c	&#99;		Small c
d	&#100;		Small d
e	&#101;		Small e
f	&#102;		Small f
g	&#103;		Small g
h	&#104;		Small h
i	&#105;		Small i
j	&#106;		Small j
k	&#107;		Small k
l	&#108;		Small l
m	&#109;		Small m
n	&#110;		Small n
o	&#111;		Small o
p	&#112;		Small p
q	&#113;		Small q
r	&#114;		Small r
s	&#115;		Small s
t	&#116;		Small t
u	&#117;		Small u
v	&#118;		Small v
w	&#119;		Small w
x	&#120;		Small x
y	&#121;		Small y
z	&#122;		Small z
{	&#123;		Left curly brace
|	&#124;		Vertical bar
}	&#125;		Right curly brace
~	&#126;		Tilde
□	&#127;		Unused
€	&#128;		Unused
			Non-breaking Space
¡	&#161;		Inverted exclamation
¢	&#162;		Cent sign
£	&#163;		Pound sterling

¤	&#164;		General currency sign
¥	&#165;		Yen sign
¦	&#166;		Broken vertical bar
§	&#167;		Section sign
¨	&#168;		Diæresis / Umlaut
©	&#169;		Copyright
ª	&#170;		Feminine ordinal
«	&#171;		Left angle quote, guillemot left
¬	&#172;		Not sign
	&#173;	&shy;	Soft hyphen
®	&#174;		Registered trademark
¯	&#175;		Macron accent
°	&#176;		Degree sign
±	&#177;		Plus or minus
²	&#178;		Superscript two
³	&#179;		Superscript three
´	&#180;		Acute accent
µ	&#181;		Micro sign
¶	&#182;		Paragraph sign
·	&#183;		Middle dot
¸	&#184;		Cedilla
¹	&#185;		Superscript one
º	&#186;		Masculine ordinal
»	&#187;		Right angle quote, guillemot right
¼	&#188;		Fraction one-fourth
½	&#189;		Fraction one-half
¾	&#190;		Fraction three-fourths
¿	&#191;		Inverted question mark
À	&#192;	&Agrave;	Capital A, grave accent
Á	&#193;	&Aacute;	Capital A, acute accent
Â	&#194;	&Acirc;	Capital A, circumflex
Ã	&#195;	&Atilde;	Capital A, tilde
Ä	&#196;	&Auml;	Capital A, diæresis / umlaut
Å	&#197;	&Aring;	Capital A, ring
Æ	&#198;	&AElig;	Capital AE ligature
Ç	&#199;	&Ccedil;	Capital C, cedilla
È	&#200;	&Egrave;	Capital E, grave accent
É	&#201;	&Eacute;	Capital E, acute accent
Ê	&#202;	&Ecirc;	Capital E, circumflex
Ë	&#203;	&Euml;	Capital E, diæresis / umlaut
Ì	&#204;	&Igrave;	Capital I, grave accent
Í	&#205;	&Iacute;	Capital I, acute accent

Î	&#206;	&Icirc;	Capital I, circumflex
Ï	&#207;	&Iuml;	Capital I, diæresis / umlaut
Ð	&#208;	&ETH;	Capital Eth, Icelandic
Ñ	&#209;	&Ntilde;	Capital N, tilde
Ò	&#210;	&Ograve;	Capital O, grave accent
Ó	&#211;	&Oacute;	Capital O, acute accent
Ô	&#212;	&Ocirc;	Capital O, circumflex
Õ	&#213;	&Otilde;	Capital O, tilde
Ö	&#214;	&Ouml;	Capital O, diæresis / umlaut
×	&#215;		Multiply sign
Ø	&#216;	&Oslash;	Capital O, slash
Ù	&#217;	&Ugrave;	Capital U, grave accent
Ú	&#218;	&Uacute;	Capital U, acute accent
Û	&#219;	&Ucirc;	Capital U, circumflex
Ü	&#220;	&Uuml;	Capital U, diæresis / umlaut
Ý	&#221;	&Yacute;	Capital Y, acute accent
Þ	&#222;	&THORN;	Capital Thorn, Icelandic
ß	&#223;	&szlig;	Small sharp s, German sz
à	&#224;	&agrave;	Small a, grave accent
á	&#225;	&aacute;	Small a, acute accent
â	&#226;	&acirc;	Small a, circumflex
ã	&#227;	&atilde;	Small a, tilde
ä	&#228;	&auml;	Small a, diæresis / umlaut
å	&#229;	&aring;	Small a, ring
æ	&#230;	&aelig;	Small ae ligature
ç	&#231;	&ccedil;	Small c, cedilla
è	&#232;	&egrave;	Small e, grave accent
é	&#233;	&eacute;	Small e, acute accent
ê	&#234;	&ecirc;	Small e, circumflex
ë	&#235;	&euml;	Small e, diæresis / umlaut
ì	&#236;	&igrave;	Small i, grave accent
í	&#237;	&iacute;	Small i, acute accent
î	&#238;	&icirc;	Small i, circumflex
ï	&#239;	&iuml;	Small i, diæresis / umlaut
ð	&#240;	&eth;	Small eth, Icelandic
ñ	&#241;	&ntilde;	Small n, tilde
ò	&#242;	&ograve;	Small o, grave accent
ó	&#243;	&oacute;	Small o, acute accent
ô	&#244;	&ocirc;	Small o, circumflex
õ	&#245;	&otilde;	Small o, tilde
ö	&#246;	&ouml;	Small o, diæresis / umlaut
÷	&#247;		Division sign
ø	&#248;	&oslash;	Small o, slash

ù	&#249;	&ugrave;	Small u, grave accent
ú	&#250;	&uacute;	Small u, acute accent
û	&#251;	&ucirc;	Small u, circumflex
ü	&#252;	&uuml;	Small u, diæresis / umlaut
ý	&#253;	&yacute;	Small y, acute accent
þ	&#254;	&thorn;	Small thorn, Icelandic
ÿ	&#255;	&yuml;	Small y, diæresis / umlaut

Source Specification

This table lists the elements and attributes supported by Internet Explorer 3.0. The table identifies the most current HTML specification that supports the element.

Tag	Attribute	Source Specification
!DOCTYPE		HTML 3.2
A		
A	HREF	HTML 2
A	NAME	HTML 2
A	REL	HTML 3.2
A	REV	HTML 3.2
A	TARGET	Netscape
A	TITLE	HTML 3.2
ADDRESS		HTML 2
APPLET		HTML 3.2
APPLET	ALIGN	HTML 3.2
APPLET	ALT	HTML 3.2
APPLET	CODE	HTML 3.2
APPLET	CODEBASE	HTML 3.2
APPLET	HEIGHT	HTML 3.2
APPLET	HSPACE	HTML 3.2
APPLET	NAME	HTML 3.2
APPLET	PARAM NAME	HTML 3.2
APPLET	VSPACE	HTML 3.2
APPLET	WIDTH	HTML 3.2
AREA		
AREA	COORDS	IE 3.0
AREA	HREF	IE 3.0
AREA	NOHREF	IE 3.0
AREA	SHAPE	IE 3.0
AREA	TARGET	Netscape
B		HTML 2.0
BASE		
BASE	HREF	HTML 2.0
BASE	TARGET	Netscape
BASEFONT		
BASEFONT	SIZE	Netscape
BGSOUND		
BGSOUND	LOOP	IE 3.0
BGSOUND	SRC	IE 3.0
BIG		HTML 3.2
BLOCKQUOTE		HTML 2.0

BODY		HTML 2.0
BODY	ALINK	HTML 2.0
BODY	BACKGROUND	HTML 3.2
BODY	BGCOLOR	Netscape
BODY	BGPROPERTIES	IE 2.0
BODY	LEFTMARGIN	IE 2.0
BODY	LINK	Netscape
BODY	TEXT	Netscape
BODY	TOPMARGIN	IE 2.0
BODY	VLINK	Netscape
BR		
BR	CLEAR	HTML 3.2
CAPTION		
CAPTION	ALIGN	HTML 3.2
CENTER		Netscape
CITE		HTML 2.0
CODE		HTML 2.0
COL		HTML 3.2
COL	ALIGN	HTML 3.2
COL	SPAN	HTML 3.2
COLGROUP		HTML 3.2
COLGROUP	ALIGN	HTML 3.2
COLGROUP	VALIGN	HTML 3.2
COMMENT		HTML 2.0
DD		HTML 2.0
DFN		HTML 2.0
DIR		HTML 2.0
DIV		HTML 3.2
DL		HTML 2.0
DT		HTML 2.0
EM		HTML 2.0
EMBED		HTML 2.0
EMBED	SRC	HTML 2.0
EMBED	WIDTH	HTML 2.0
EMBED	HEIGHT	HTML 2.0
EMBED	PALETTE	HTML 2.0
EMBED	NAME	HTML 2.0
EMBED	OPTIONAL PARAM=	HTML 2.0
FONT		
FONT	COLOR	IE 3.0
FONT	FACE	IE 3.0
FONT	SIZE	Netscape
FORM		

FORM	ACTION	HTML 2.0
FORM	METHOD	HTML 2.0
FORM	TARGET	Netscape
FRAME		
FRAME	FRAMEBORDER	IE 3.0
FRAME	MARGINHEIGHT	Netscape
FRAME	MARGINWIDTH	Netscape
FRAME	NAME	Netscape
FRAME	NORESIZE	Netscape
FRAME	SCROLLING	Netscape
FRAME	SRC	Netscape
FRAMESET		
FRAMESET	COLS	Netscape
FRAMESET	FRAMEBORDER	IE 3.0
FRAMESET	FRAMESPACING	IE 3.0
FRAMESET	ROWS	Netscape
H*n*		HTML 2
H*n*	ALIGN	HTML 3.2
HR		HTML 2
HR	ALIGN	HTML 3.2
HR	COLOR	IE 3.0
HR	NOSHADE	Netscape
HR	SIZE	Netscape
HR	WIDTH	Netscape
HTML		HTML 2.0
I		HTML 2.0
IMG		
IMG	ALIGN	HTML 3.2
IMG	ALT	HTML 2
IMG	BORDER	HTML 3.2
IMG	CONTROLS	IE 3.0
IMG	DYNSRC	IE 3.0
IMG	HEIGHT	HTML 3.2
IMG	HSPACE	HTML 3.2
IMG	ISMAP	HTML 3.2
IMG	LOOP	IE 3.0
IMG	SRC	HTML 2
IMG	START	IE 3.0
IMG	USEMAP	IE 3.0
IMG	VSPACE	HTML 3.2
IMG	WIDTH	HTML 3.2
INPUT		

INPUT	ALIGN	HTML 2.0
INPUT	CHECKED	HTML 2.0
INPUT	MAXLENGTH	HTML 2.0
INPUT	NAME	HTML 2.0
INPUT	SIZE	HTML 2.0
INPUT	SRC	HTML 2.0
INPUT	TYPE	HTML 2.0
INPUT	TYPE=CHECKBOX	HTML 2.0
INPUT	TYPE=HIDDEN	HTML 2.0
INPUT	TYPE=IMAGE	HTML 2.0
INPUT	TYPE=PASSWORD	HTML 2.0
INPUT	TYPE=RADIO	HTML 2.0
INPUT	TYPE=RESET	HTML 2.0
INPUT	TYPE=SUBMIT	HTML 2.0
INPUT	TYPE=TEXT	HTML 2.0
INPUT	TYPE=TEXTAREA	HTML 2.0
INPUT	VALUE	HTML 2.0
ISINDEX		HTML 2.0
ISINDEX	ACTION	Netscape
ISINDEX	PROMPT	HTML 3.2
KBD		HTML 2.0
LI		HTML 2.0
LI	TYPE	Netscape
LI	VALUE	Netscape
LINK		IE 3.0
LISTING		HTML 2.0
MAP		
MAP	NAME	IE 3.0
MARQUEE		IE 3.0
MARQUEE	ALIGN	IE 3.0
MARQUEE	BEHAVIOR	IE 3.0
MARQUEE	BGCOLOR	IE 3.0
MARQUEE	DIRECTION	IE 3.0
MARQUEE	HEIGHT	IE 3.0
MARQUEE	HSPACE	IE 3.0
MARQUEE	LOOP	IE 3.0
MARQUEE	SCROLLAMOUNT	IE 3.0
MARQUEE	SCROLLDELAY	IE 3.0
MARQUEE	VSPACE	IE 3.0
MARQUEE	WIDTH	IE 3.0
MENU		HTML 2
META		
META	CONTENT	Netscape

META	HTTP-EQUIV	Netscape
META	NAME	HTML 3.2
NOBR		Netscape
NOFRAMES		Netscape
OL		HTML 2
OL	START	Netscape
OL	TYPE	Netscape
OPTION		HTML 2
OPTION	SELECTED	HTML 2
OPTION	VALUE	HTML 2
P		HTML 2
P	ALIGN	HTML 3.2
PLAINTEXT		HTML 2
PRE		HTML 2
S		HTML 2
SAMP		HTML 2
SELECT		HTML 2
SELECT	MULTIPLE	HTML 2
SELECT	NAME	HTML 2
SELECT	SIZE	HTML 2
SMALL		HTML 3.2
SPAN		HTML 3.2
STRIKE		HTML 2
STRONG		HTML 2
SUB		HTML 3.2
SUP		HTML 3.2
TABLE		
TABLE	ALIGN	HTML 3.2
TABLE	BACKGROUND	IE 3.0
TABLE	BGCOLOR	IE 3.0
TABLE	BORDERCOLOR	IE 3.0
TABLE	BORDERCOLORDARK	IE 3.0
TABLE	BORDERCOLORLIGHT	IE 3.0
TABLE	FRAME	HTML 3.2
TABLE	RULES	HTML 3.2
TBODY		HTML 3.2
TD		
TD	ALIGN	HTML 3.2
TD	BGCOLOR	IE 3.0
TD	BORDERCOLOR	IE 3.0
TD	BORDERCOLORDARK	IE 3.0
TD	BORDERCOLORLIGHT	IE 3.0
TD	VALIGN	IE 3.0

TFOOT		HTML 3.2
TH		
TH	ALIGN	HTML 3
TH	BGCOLOR	IE 3.0
TH	BORDERCOLOR	IE 3.0
TH	BORDERCOLORDARK	IE 3.0
TH	BORDERCOLORLIGHT	IE 3.0
TH	VALIGN	IE 3.0
THEAD		HTML 3
TITLE		HTML 2
TR		
TR	ALIGN	HTML 3
TR	BGCOLOR	IE 3.0
TR	BORDERCOLOR	IE 3.0
TR	BORDERCOLORDARK	IE 3.0
TR	BORDERCOLORLIGHT	IE 3.0
TR	VALIGN	IE 3.0
TT		HTML 2
U		HTML 2
UL		HTML 2
VAR		HTML 2
WBR		Netscape
XMP		HTML 2

HTML Reference

Tag	Description
!--...--	Comments. Any text between tags will not display in the browser.
!DOCTYPE	Describes the HTML version used in the current document.
A	Stands for *anchor*. HREF= attribute creates hyperlinks. NAME= attribute creates a named references.
ADDRESS	Specifies a mailing address.
APPLET	Embeds a Java applet. See OBJECT.
AREA	Specifies the shape of a "hot spot" in a client-side image map.
B	Changes text to bold. See STRONG.
BASE	Specifies a document's URL.
BASEFONT	Sets the base font value.
BGSOUND	Adds background sounds that play on initial load.
BIG	Enlarges the font size.
BLOCKQUOTE	Sets apart a quotation in text.
BODY	Specifies the beginning and the end of document body. See HEAD.
BR	Inserts a line break.
CAPTION	Specifies a caption for a table. Valid only within the TABLE element.
CENTER	Centers text and images.
CITE	Indicates a citation. Used to present a book, paper, or other published source material.
CODE	Presents a code sample.
COL	Sets the properties of a column.
COLGROUP	Sets the properties of one or more column as a group.
COMMENT	Indicates a comment. Text in a comment element does not display in a browser.
DD	Specifies definition data. Used to format the text for a definition. See DL, DT.
DFN	Specifies a definition. Formats a defined term.
DIR	Denotes a directory list.
DIV	Sets a document division. Groups related elements together within a document.
DL	Denotes a definition list. Used for a list of defined terms. See DT, DD.
DT	Specifies a defintion term. Used to format the defined term. See DL, DD.
EM	Emphasizes text, usually by rendering text in italics.
EMBED	Indicates an embedded objects. See OBJECT.
FONT	Formats the font style, size and color.
FORM	Denotes a form with which users enter data. See INPUT for a list of form elements.
FRAME	Defines independent windows, or frames, within a page. See FRAMESET.

FRAMESET	Defines layout for frames within a page. See FRAME.
Hn	Renders text in heading style, usually with a larger font than the body text. N is a value from 1 to 6.
HEAD	Marks the HTML document heading.
HR	Draws a horizontal rule. Used to separate sections.
HTML	Denotes the file is an HTML document.
I	Renders text in italics.
IMG	Inserts a graphic file.
INPUT	Specifies a form control such as a checkbox or radio button. See FORM.
ISINDEX	Indicates the presence of an index.
KBD	Indicates text to be entered at a keyboard. Appears in fixed-width and boldface type.
LI	Denotes an item in a list. Adds special character or number depending on use. See UL, OL.
LINK	Establishes the relationship between documents. Appears only in HEAD element.
LISTING	Renders text in fixed-width type.
MAP	Specifies a collection of hot spots for a client-side image map.
MARQUEE	Displays text in a scrolling marquee.
MENU	Denotes a list of items.
META	Provides information about the document. Used for client pull, also by some search engines for indexxing.
NOBR	Turns off line breaking.
NOFRAMES	Indicates content viewable only by browsers that do not support frames.
OBJECT	Inserts an OLE Control.
OL	Specifies an ordered list. Each item has a number or letter reference. See UL, LI.
OPTION	Denotes one choice in a list box.
P	Inserts a paragraph break and denotes a new paragraph.
PARAM	Sets object properties.
PLAINTEXT	Renders text in fixed-width type without processing elements.
PRE	Displays text exactly as typed — with all line breaks and spacing.
S	Renders text in strikethrough type.
SAMP	Specifies sample text. See CODE.
SELECT	Denotes a list box or dropdown list.
SMALL	Decreases the font size.
SPAN	Use to apply style information to the enclosed text.
STRIKE	Renders text in strikethrough type. See S.
STRONG	Emphasizes text, usually with boldface. See B.
SUB	Renders text in subscript.
SUP	Renders text in superscript.
TABLE	Creates a table. See TH, TR, and TD to learn how to define rows and columns.

TBODY	Defines the table body.
TD	Creates a cell in a table.
TFOOT	Defines the table footer.
TH	Creates a row or column heading in a table.
THEAD	Defines the table header.
TEXTAREA	Creates a box in which a user can enter and edit text.
TITLE	Specifies a document title. Appears in the browser title bar.
TR	Creates a row in a table.
TT	Denotes teletype. Displays text in fixed-width type.
U	Renders text underlined.
UL	Formats lines of text as a bulleted list. See LI.
VAR	Indicates placeholder text for a variable. Displays text in a small fixed-width font.
WBR	Inserts a soft linebreak in a block of NOBR text.
XMP	Indicates example text. Displays text in fixed-width font.

!--...--

Specifies that enclosed text is an author's comment. Any text between the lines will not print. You may include multiple lines of text between the start-tag and end-tag.

<!-- ... -->

Example

```
<!-- This line text, enclosed in an HTML page, will not print.
    This line of text, will not print                      -->
```

!DOCTYPE

Specifies the version of HTML used in the document. !DOCTYPE is the first element in any HTML document. !DOCTYPE is a required element for any HTML 3.2 compliant document.

<!DOCTYPE>

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```

A

Stands for anchor. The A and /A tags enclose text or graphics. The properties of elements that can follow A are applied to the bracketed text or graphic. A is used to attach a hyperlink to text or graphics using the HREF= attribute. A is used to specify text or graphics as a named reference using the NAME= attribute. Anchors cannot be nested.

<**A**
HREF=*reference*
NAME=*name*
REL=*relationship*
REV=*revision*
TARGET=*window*
TITLE=*title*
>

Parameters

HREF=*reference*
Specifies either a destination address or a destination file. A destination address must be in URL format. A destination file must name a file and be in the format of the given file system. If no path or domain name is specified, the file is searched for in the same location as the current document.

NAME=*name*
Specifies a named reference within an HTML document. The name can be referenced by its document and external documents in a hyperlink by prefacing it with the pound sign, #.

REL=*relationship*
Specifies a relative relationship.

REV=*revision*
Specifies the revision number.

TARGET=*window*
Specifies to load the link into the targeted window. This attribute can be used with a frameset where a frame has been named in the FRAME element. *Window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alpha-numeric character to be valid, except for the four target windows detailed below.
_blank	Specifies to load the link into a new blank window. This window is not named.
_parent	Specifies to load the link into the immediate parent of the document the link is in.
_self	Specifies to load the link into the same window the link was clicked in.
_top	Specifies to load the link into the full body of the window.

TITLE=*name*
Specifies the title that appears when the hyperlink is selected.

Example

```
<A HREF="http://www.microsoft.com"> This is a link to Microsoft.</A>
```

```
<A HREF="home.htm">This is a link to a file called Home.htm in the same
directory as this page.</A>

<A TARGET="viewer" HREF="sample.htm">Click here to load the link into
"viewer" window.</A>
```

ADDRESS

Specifies the mailing address. This element typically is used at the bottom of a document. Text is displayed in italics.

<ADDRESS>

Example

```
<ADDRESS>This text will be in italics.</ADDRESS>
```

APPLET

Embeds a Java applet in an HTML document.

<**APPLET ALIGN=***LEFT*|**RIGHT**|**CENTER ALT=***alternateText* **CODEBASE=***codebaseURL* **CODE=***appletFile* **HEIGHT=***pixels* **HSPACE=***pixels* > **NAME=***appletInstanceName* [< **PARAM NAME =** *AttributeName* >] **WIDTH=***pixels* **VSPACE=***pixels* <**/APPLET>**

Parameters

ALIGN=*alignment*
Alignment of object to text.

ALT=*alternateAppletText*
Alternate text for text-only browsers or browsers that do not support Java.

CODE=*appletFile*
The name of the Java applet.

CODEBASE*codebaseURL*
The base URL of the applet. The directory in which the applet is located.

HEIGHT=*pixels*
Initial height of the applet display area.

HSPACE=*pixels*
Horizontal space.

NAME=*appletInstanceName*
Use NAME to identify an applet to other applets within the HTML page.

PARAM NAME=*AttributeName*
Use NAME= to pass applet-specific arguments in from an HTML page.

VSPACE=*pixels*
Space in pixels above the applet.

WIDTH=*pixels*
Initial width of the applet display area.

AREA

Specifies the shape of a "hot spot" in a client-side image map.

<AREA
COORDS=*coords*
SHAPE=*shape-type*
NOHREF
HREF=*url*
TARGET=*window* >

Parameters

COORDS=*coords*
Specifies coordinates that define the hot spot's shape.

HREF=*url*
Specifies the destination of the hot spot.

NOHREF
Indicates that clicks in this region should cause no action.

SHAPE=*shape-type*
Denotes the type of shape. The *shape-type* can be one of these values:

RECT	Rectangle. Takes four coordinates: *x1*, *y1*, *x2*, and *y2*.
RECTANGLE	Rectangle. Takes four coordinates: *x1*, *y1*, *x2*, and *y2*.
CIRC	Circle. Takes three coordinates: *centerx*, *centery*, and *radius*.
CIRCLE	Circle. Takes three coordinates: *centerx*, *centery*, and *radius*.
POLY	Polygon. Takes three or more pairs of coordinates denoting a polygonal region.
POLYGON	Polygon. Takes three or more pairs of coordinates denoting a polygonal region.

TARGET=*window*
Specifies to load the link into the targeted window. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alphanumeric character to be valid, except for the four target windows detailed below.
_blank	Specifies to load the link into a new blank window. This window is not named.
_parent	Specifies to load the link into the immediate parent of the document the link is in.
_self	Specifies to load the link into the same window the link was clicked in.
_top	Specifies to load the link into the full body of the window.

Examples

```
<AREA SHAPE="RECT" COORDS="50, 25, 150, 125" HREF="http://www.sample.com">

<AREA SHAPE="RECT" COORDS="50, 25, 150, 125" NOHREF>
```

```
<AREA TARGET="viewer" HREF="sample.htm" SHAPE="CIRCLE" COORDS="50, 25, 150,
125">
```

B

Renders text in bold.

<B>

Example

```
<B>Displayed in a bold typeface.</B>
```

BASE

Specifies the document's URL.

<BASE
HREF=*url*
TARGET=*window***>**

Parameters

HREF=*url*

Specifies the document's full URL in case the document gets read out of context and the reader wants to refer to the original.

TARGET=*window*

Specifies to load all of the links on the page into the targeted window. This can be overridden by specifying a different target attribute for a specific link. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alpha-numeric character to be valid, except for the four target windows detailed below.
_blank	Specifies to load the link into a new blank window. This window is not named.
_parent	Specifies to load the link into the immediate parent of the document the link is in.
_self	Specifies to load the link into the same window the link was clicked in.
_top	Specifies to load the link into the full body of the window.

Examples

```
<BASE HREF="http:// www.sample.com/hello.htm">

<BASE HREF="http:// www.sample.com/hello.htm" TARGET="viewer">
```

BASEFONT

Sets the base font value.

<BASEFONT
SIZE=*n*>

Parameters

SIZE=*n*

Sets the base font size. The *n* can be between 1 and 7 inclusive; default is 3; 7 is largest. Throughout the document, relative font size settings (for example, <FONT SIZE=+3>) are set according to this.

Example

```
<BASEFONT SIZE=3> This sets the base font size to 3.
<FONT SIZE=+4> Now the font size is 7.
<FONT SIZE=-1> Now the font size is 2.
```

BGSOUND

Adds background sounds or "soundtracks" to a page. Sounds can either be samples (WAV or AU format) or MIDI format.

<**BGSOUND**
SRC=*url*
LOOP=*n* >

Parameters

SRC=*url*
Specifies the address of a sound to be displayed.

LOOP=*n*
Specifies how many times a sound will loop when activated. If *n*=-1, or if *LOOP*=INFINITE is specified, it will loop indefinitely.

BIG

Makes text one size larger.

<BIG>

Example

```
<BIG>This text is larger<./BIG>
```

BLOCKQUOTE

Indents both left and right margins. Used to set apart quotations in text.

<BLOCKQUOTE>

Example

```
<P>He said,
<BLOCKQUOTE>"Hi there!" </BLOCKQUOTE>
```

BODY

Specifies the beginning and end of the document body. This element also allows you to set the background image, the background color, the link colors, and the top and left margins of the page.

<**BODY**
BACKGROUND=*url*
BGCOLOR=*color*
BGPROPERTIES=FIXED
LINK=*color*
TEXT=*color*
TOPMARGIN=*n*
VLINK=*color*>

Parameters

BACKGROUND=*url*
Specifies a background picture. The picture is tiled behind the text and graphics on the page.

BGCOLOR=*color*
Sets the background color of the page. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BGPROPERTIES=FIXED
Specifies a watermark, which is a background picture that does not scroll.

LEFTMARGIN=*n*
Specifies the left margin for the entire body of the page and overrides the default margin. If set to zero, the left margin will be exactly on the left edge.

LINK=*color*
Sets the color of hyperlinks that have not yet been visited. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

TEXT=*color*
Sets the color of text on the page. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

TOPMARGIN=*n*
Specifies the margin for the top of the page and overrides the default margin. If set to zero, the top margin will be on the precise top edge.

VLINK=*color* or *colorname*
Sets the color of hyperlinks that have already been visited. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

Examples

The HTML used to insert the background image of this page is:

```
<BODY BACKGROUND="/ie/images/watermrk.gif" BGPROPERTIES=FIXED
BGCOLOR=#FFFFFF TEXT=#000000 LINK=#ff6600 VLINK=#330099>

<HTML> <BODY>Here's a Web page!</BODY></HTML>
```

BR

Inserts a line break.

<BR
CLEAR=align-type>

Parameters

CLEAR=*align-type*

Inserts vertical space so that the next text displayed will be past left- or right-aligned "floating" images. The *align-type* can be one of these values:

LEFT	Inserts space so that the next text appears aligned with the left margin directly below a left-aligned floating image.
RIGHT	Inserts space so that the next text appears aligned with the right margin directly below a right-aligned floating image.
ALL	Places the next text past all floating images.

CAPTION

Specifies a caption for a table.

<CAPTION
ALIGN=*align-type***>**

Parameters

ALIGN=*align-type*
Sets the alignment of the caption within the table. The *align-type* can be LEFT, RIGHT, TOP or BOTTOM. By default, the caption is centered and at the bottom of the table.

Remarks

This element is valid only within the TABLE element. The end-tag is required.

Example

```
<TABLE>
<CAPTION ALIGN=BOTTOM>
This caption will appear centered below the table.
</CAPTION>
<TR>
     ....
</TR>
</TABLE>
```

CENTER

Centers text and images.

Example

```
<CENTER>Hi there!</CENTER>
```

CITE

Indicates a citation. Refers to a book, paper, or other published source material.

<CITE>

Example

```
<CITE>Book Title.</CITE>
```

CODE

Specifies a code sample. Renders text in a small font. (If no font face is specified, the font used is fixed-width.)

<CODE>

Example

```
<CODE>Here is some text in a small fixed-width font.</CODE>
```

COL

Sets the properties of one or more columns. Use this element in conjunction with a COLGROUP element to set the properties of a column within a group of columns.

<**COL**
ALIGN=*align-type*
SPAN=*n*>

Parameters

ALIGN=*align-type*
Specifies the text alignment in cells within the column. The *align-type* can be CENTER, LEFT, or RIGHT.

SPAN=*n*
Sets the number of consecutive columns for which the properties are set.

Remarks

This element is valid only within a table. The end-tag is not required and not recommended.

The properties specified by the COL element always override the properties specified by the preceding COLGROUP element.

Example

```
<TABLE>
<COLGROUP>
    <COL ALIGN=RIGHT>
    <COL ALIGN=LEFT>
<COLGROUP>
    <COL ALIGN=CENTER>
<TBODY>
    <TR>
    <TD>This is the first column in the group and is right-aligned.</TD>
    <TD>This is the second column in the group and is left-aligned.</TD>
    <TD>This column is in a new group and is centered.</TD>
    </TR>
</TABLE>
```

COLGROUP

Sets the properties of one or more columns.

<**COLGROUP**
ALIGN=*align-type*
SPAN=*n*>

Parameters

ALIGN=*align-type*
Specifies the alignment of text in the cells in the column(s). The *align-type* can CENTER, LEFT or RIGHT.

SPAN=*n*
Sets the number of consecutive columns that are in the group and for which the properties are set.

Remarks

This element is valid only within a table. The end-tag is not required and not recommended.

If the columns in a group of columns require varying properties, use COLGROUP in conjunction with one or more COL elements to individually set the properties for the columns.

This element affects how rules are drawn within a table when groups are specified with the RULES= attribute in the table element. In this case, vertical rules are drawn between column groups rather than between individual columns.

Example

```
<TABLE>
<COLGROUP ALIGN=RIGHT>
<COLGROUP SPAN=2 ALIGN=LEFT>
<TBODY>
    <TR>
    <TD>This column is in the first group and is right-aligned</TD>
    <TD>This column is in the second group and is left-aligned</TD>
    <TD>This column is in the second group and is left-aligned</TD>
    </TR>
</TABLE>
```

COMMENT

Indicates a comment. The text between the elements is ignored, unless it contains HTML code.

Example

```
<COMMENT>This won't be printed.</COMMENT>
```

DD

Specifies a definition in a definition list. Indicates that the text is a definition of a term, and should therefore be displayed in the right-hand column of a definition list.

<DD>

Example

```
<DL> <DT>Cat<DD>A furry cute animal that purrs and likes milk.
<DT>Lizard<DD>A weird desert animal with a long tongue.</DL>
```

DFN

Specifies a definition. Formats a term for its first appearance in a document.

<DFN>

Example

```
<DFN>HTML stands for hypertext markup language.</DFN>
```

DIR

Denotes a directory list. Specifies that the following block consists of individual items, each beginning with an LI element and none containing more than 20 characters, which should be displayed in columns.

<DIR>

Example

```
<DIR> <LI>Art
<LI>History
<LI>Literature
<LI>Sports
<LI>Entertainment
<LI>Science
</DIR>
```

DIV

Sets document division. Groups related elements together.

<DIV ALIGN=*align-type* **</DIV>**

Parameters

ALIGN=*align-type*
Sets alignment of block elements within DIV elements. The *align-type* can be LEFT, CENTER, or RIGHT. LEFT is the default value.

Example

```
<DIV>
This text represents a section.
</DIV>
<ex><DIV ALIGN=CENTER>
This text represents another section.
</DIV>
```

DL

Specifies that the following block is a definition list: that is, an automatically formatted two-column list with terms on the left and their definitions on the right.

<DL>

Example

```
<DL>
<DT>Cat
<DD>A furry cute animal that purrs and likes milk.
<DT>Lizard
<DD>A weird desert animal with a long tongue.
</DL>
```

DT

Specifies a term in a definition list. Indicates that the text is a term to be defined, and should therefore be displayed in the left-hand column of a definition list.

<DT>

Example

```
<DL> <DT>Cat<DD>A furry cute animal that purrs and likes milk.
<DT>Lizard<DD>A weird desert animal with a long tongue.</DL>
```

EM

Emphasizes text, usually by rendering it in italics.

Example

```
<EM>This text will be in italics.</EM>
```

EMBED

Indicates an embedded object. OBJECT is the preferred element for inserting objects, but EMBED is included for backward compatibility with earlier HTML documents. See OBJECT.

<**EMBED**
HEIGHT=*size* **of object**
NAME=*programmatic* **name**
OPTIONAL PARAM="value" ... OPTIONAL PARAM=
PALETTE=*foreground* | **background**
SRC=*data* **to object**
WIDTH=*size* **of object>**

Parameters

HEIGHT=*size of object*
The height, in pixels, of the object on the page.

NAME=*programmatic name*
The name used by other objects or elements to refer to this object.

OPTIONAL PARAM=*value*
Specifies any parameters that are specific to the object.

PALETTE=*foreground* | *background*;
Sets the color palette to the foreground or background color.

SRC=*data to object*
The name of any source data input to the object.

WIDTH=*size of object*
The width of the object, in pixels, on the page.

Example

```
<EMBED SRC=&amp;quot;MyMovie.AVI&amp;quot; WIDTH=100 HEIGHT=250 AUTOSTART=TRUE
PLAYBACK=FALSE&amp;gt;</code></font></font></pre>
```

FONT

Sets the font, size, and color of text.

<**FONT**
SIZE=*n*
FACE=*name*
COLOR=*color*>

Parameters

COLOR=*color*
Sets font color. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

FACE="*name* [,*name2*[,*name3*]]"
Sets the font. A list of font names can be specified. If the first font is available on the system, it will be used, otherwise the second will be tried, and so on. If none are available, a default font will be used.

SIZE=*n*
Specifies font size between 1 and 7 (7 is largest). A plus or minus before the number indicates a size relative to the current BASEFONT setting. Relative font sizes are not cumulative, so putting two <FONT SIZE="+1"> elements in a row does not result in the font size being increased by 2.

FORM

Denotes a form.

<**FORM**
ACTION=*url*
METHOD=*get-post*
TARGET=*window*>

Parameters

ACTION=*url*

Specifies the address to be used to carry out the action of the form. If none is specified, the base URL of the document is used.

METHOD=*get-post*

Indicates how the form data should be sent to the server. The get-post can be one of these values:

GET	Appends the arguments to the action URL and open it as if it were an anchor.
POST	Sends the data via an HTTP post transaction.

TARGET=*window*

Specifies to load the results of the form submission into the targeted window. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alpha-numeric character to be valid, except for the four target windows detailed below.
_blank	Specifies to load the link into a new blank window. This window is not named.
_parent	Specifies to load the link into the immediate parent of the document the link is in.
_self	Specifies to load the link into the same window the link was clicked in.
_top	Specifies to load the link into the full body of the window.

Example

```
<FORM TARGET="viewer" ACTION="http://www.sample.com/bin/search">
    ...
</FORM>
```

FRAME

Defines a single frame in a frameset. There is no matching end-tag as this is not a container.

<**FRAME**
ALIGN=*align-type*
FRAMEBORDER=*1*|**0**
MARGINHEIGHT=*height*
MARGINWIDTH=*width*
NAME=*name*
SCROLLING=*yes*|**no**
SRC=*address*

Parameters

ALIGN=*align-type*
Sets the alignment of the frame or of the surrounding text. The *align-type* can be one of these values:

TOP	Surrounding text is aligned with the top of the frame.
MIDDLE	Surrounding text is aligned with the middle of the frame.
BOTTOM	Surrounding text is aligned with the bottom of the frame.
LEFT	The frame is drawn as a left-flush "floating frame," and text flows around it.
RIGHT	The frame is drawn as a right-flush "floating frame," and text flows around it.

FRAMEBORDER=*0*|*1*
Renders a 3-D edge border around the frame. 1 (default) inserts a border. 0 displays no border.

MARGINHEIGHT=*height*
Controls the margin height for the frame in pixels.

MARGINWIDTH=*width*
Controls the margin width for the frame in pixels.

NAME=*name*
Provides a target name for the frame.

NORESIZE
Prevents the user from resizing the frame.

SCROLLING=*yes*|*no*
Creates a scrolling frame.

SRC=*address*
Displays the source file for the frame.

Example

```
<FRAME FRAMEBORDER=0 SCROLLING=NO SRC="sample.htm">
```

FRAMESET

This is a container which hosts the FRAME, **FRAMESET**, and NOFRAMES elements.

<**FRAMESET**
COLS=*col-widths*
FRAMEBORDER=*1*|**0**
FRAMESPACING=*spacing*
ROWS=*row-heights*

Parameters

COLS=*col-widths*
Creates a frame document with columns. You can specify the column dimensions by percentage (%), pixels, or a relative size (*).

FRAMEBORDER=*1*|*0*
Provides the option to display or not display a 3-D border for a frame. 1 (default) sets a frame border. 0 displays no border.

FRAMESPACING=*spacing*
Creates additional space between frames in pixels.

ROWS=*row-heights*
Creates a frame document with rows. You can specify the row dimensions by percentage (%), pixels, or a relative size (*).

Remarks

The **FRAMEBORDER=** and **FRAMESPACING=** attributes are inherited from any containing **FRAMESET** element, which means you only need to set the attribute on the single, outermost **FRAMESET** tag to affect all FRAME tags on that page.

Example

```
<FRAMESET SCROLLING=YES COLS="25%, 50%, *">
    <FRAME SRC="contents.htm">
    <FRAME SRC="info.htm">
    <FRAME SCROLLING=NO SRC="graphic.htm">
</FRAMESET>
```

Hn

Renders text in heading style. Use H1 through H7 to specify different sizes and styles of heading.

<**H***n*
ALIGN=*align-type*>

Parameters

n
Sets the section level. Integer from 1 to 6.

ALIGN=*align-type*
Sets alignment of header text. *Align-type* can be CENTER, LEFT or RIGHT. LEFT is default.

Remarks

The end-tag is required.

Example

```
<H1>Welcome to Internet Explorer!</H1>
```

HEAD

Marks the HTML document heading.

<HEAD>

Remarks

The end-tag is not required.

Example

```
<HEAD>
<TITLE>A Simple Document</TITLE>
</HEAD>
```

HR

Draws a horizontal rule.

<HR
ALIGN=*align-type*
COLOR=*color*
NOSHADE
SIZE=*n*
WIDTH=*n***>**

Parameters

ALIGN=*align-type*
Draws the rule left-aligned, right-aligned, or centered. The *align-type* can be LEFT, RIGHT, or CENTER.

COLOR=*color*
Sets the color of the rule. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

NOSHADE
Draws the rule without 3-D shading.

SIZE=*n*
Sets the height of the rule in pixels.

WIDTH=*n*
Sets the width of the rule either in pixels or as a percentage of window width. To specify a percentage, the *n* must end with the percent (%) sign.

Example

```
<HR SIZE=5 WIDTH=80% NOSHADE>
```

HTML

Denotes the file as an HTML document.

This element has no attributes.

<HTML>

Example

```
<HTML>
<BODY>
<P>This is an HTML document.
</BODY>
</HTML>
```

I

Renders text in italics.

<I>

Example

```
<I>This text will be in italics.</I>
```

IMG

Inserts an image.

<**IMG**
ALIGN=*align-type*
ALT=*text*
BORDER=*n*
CONTROLS
DYNSRC=*url*
HEIGHT=*n*
HSPACE=*n*
ISMAP
LOOP=*n*
SRC=*address*
START=*start-event*
USEMAP=*map-name*
VSPACE=*n*
WIDTH=*n*>

Parameters

ALIGN=*align-type*

Sets the alignment of the image or of the surrounding text. The *align-type* can be one of these values:

TOP	Surrounding text is aligned with the top of the image.
MIDDLE	Surrounding text is aligned with the middle of the image.
BOTTOM	Surrounding text is aligned with the bottom of the image.
LEFT	The picture is drawn as a left-flush "floating image," and text flows around it.
RIGHT	The picture is drawn as a right-flush "floating image," and text flows around it.

ALT=*text*

Specifies text that will be displayed in place of the picture if Show Pictures is turned off.

BORDER=*n*

Specifies the size of a border to be drawn around the image. If the image is a hyperlink, the border is drawn in the appropriate hyperlink color. If the image is not a hyperlink, the border is invisible.

CONTROLS

If a video clip is present, a set of controls is displayed under the clip.

DYNSRC=*url*

Specifies the address of a video clip or VRML world to be displayed in the window. Stands for Dynamic Source.

HEIGHT=*n*

Along with **WIDTH=**, specifies the size at which the picture is drawn. If the picture's actual dimensions differ from those specified, the picture is stretched to match what's specified. Internet Explorer also uses this to draw a placeholder of appropriate size for the picture before it's loaded.

HSPACE=*n*

Along with **VSPACE=**, specifies margins for the image. Similar to **BORDER=**, except the margins are not painted with color when the image is a hyperlink.

ISMAP
Identifies the picture as a server-side image map. Clicking the picture transmits the coordinates of the click back to the server, triggering a jump to another page.

LOOP=*n*
Specifies how many times a video clip will loop when activated. If *n*=-1, or if LOOP=INFINITE is specified, it will loop indefinitely.

SRC=*address*
Specifies the address of the picture to insert.

START=*start-event*
Specifies when the file specified by the **DYNSRC=** attribute should start playing. The *start-event* can be one of these values:

FILEOPEN	Start playing as soon as the file is done opening. This is the default.
MOUSEOVER	Start playing when the user moves the mouse pointer over the animation.

Both values can be set but must be separated with a comma.

USEMAP=*map-name*
Identifies the picture as a client-side image map and specifies a MAP to use for acting on the user's clicks.

VSPACE=*n*
Along with **HSPACE=**, specifies margins for the image. Similar to **BORDER=**, except the margins are not painted with color when the image is a hyperlink.

WIDTH=*n*
Along with **HEIGHT=**, specifies the size at which the picture is drawn. If the picture's actual dimensions differ from those specified, the picture is stretched to match what's specified. Internet Explorer also uses this to draw a placeholder of appropriate size for the picture before it's loaded.

INPUT

Specifies a form control.

<**INPUT**
ALIGN=*align-type*
[**CHECKED**|]
MAXLENGTH=*length*
NAME=*name*
SIZE=*size*
SRC=*address*
TYPE=*type*
VALUE=*value*>

Parameters

ALIGN=*align-type*
Used when TYPE=IMAGE. Specifies how the next line of text will be aligned with the image. The *align-type* can be TOP, MIDDLE, or BOTTOM.

CHECKED
Include CHECKED to set a checkbox or radio button to "selected" when the form first loads.

MAXLENGTH=*length*
Indicates the maximum number of characters that can be entered into a text control.

NAME=*name*
Specifies the name of the control.

SIZE=*size*
Specifies the size of the control (in characters). For TEXTAREA-type controls, both height and width can be specified, using this format: "*width*,*height*".

SRC=*address*
Used when TYPE=IMAGE. Specifies the address of the image to be used.

TYPE=*type*
Specifies what type of control to use. The *type* can be one of these:

CHECKBOX	Used for simple Boolean attributes or for attributes that can take multiple values at the same time. It is represented by a number of check box fields, each of which has the same name. Each selected check box generates a separate name/value pair in the submitted data, even if this results in duplicate names. The default value for check boxes is "on."
HIDDEN	No field is presented to the user, but the content of the field is sent with the submitted form. This value can be used to transmit state information about client/server interaction.
IMAGE	An image field which you can click, causing the form to be immediately submitted. The coordinates of the selected point are measured in pixel units from the upper-left corner of the image, and are returned (along with the other contents of the form) in two name/value pairs. The x-coordinate is submitted under the name of the field with ".x" appended, and the y-coordinate is submitted under the name of the field with ".y" appended. Any VALUE attribute is ignored. The image itself is specified by the SRC attribute, exactly as for the Image element.

PASSWORD	The same as the TEXT attribute, except that text is not displayed as the user enters it.
RADIO	Used for attributes that accept a single value from a set of alternatives. Each radio-button field in the group should be given the same name. Only the selected radio button in the group generates a name/value pair in the submitted data. Radio buttons require an explicit VALUE attribute.
RESET	A button that when clicked resets the form's fields to their specified initial values. The label to be displayed on the button can be specified just as for the SUBMIT button.
SUBMIT	A button that when clicked submits the form. You can use the VALUE attribute to provide a non-editable label to be displayed on the button. The default label is application-specific. If a SUBMIT button is clicked in order to submit the form, and that button has a NAME attribute specified, then that button contributes a name/value pair to the submitted data. Otherwise, a SUBMIT button makes no contribution to the submitted data.
TEXT	Used for a single-line text-entry field. Use in conjunction with the SIZE and MAXLENGTH attributes.

This default control type is TEXT.

VALUE=*value*

For textual/numerical controls, specifies the default value of the control. For Boolean controls, specifies the value to be returned when the control is turned on.

Example

```
<FORM ACTION="http://intranet/survey" METHOD=POST>
<P>Name
<BR><INPUT NAME="CONTROL1" TYPE=TEXT VALUE="Your Name">
<P>Password
<BR><INPUT TYPE="PASSWORD" NAME="CONTROL2">
<P>Color
<BR><INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="0" CHECKED>Red
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="1">Green
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="2">Blue
<P>Comments
<BR><INPUT TYPE="TEXTAREA" NAME="CONTROL4" SIZE="20,5" MAXLENGTH="250">
<P><INPUT NAME="CONTROL5" TYPE=CHECKBOX CHECKED>Send receipt
<P><INPUT TYPE="SUBMIT" VALUE="OK"><INPUT TYPE="RESET" VALUE="Reset">
</FORM>
```

ISINDEX

Indicates the presence of a searchable index.

<**ISINDEX**
ACTION=*url*
PROMPT=*prompt-text*>

Parameters

ACTION=*url*
Specifies the gateway program to which the string in the text box should be passed.

PROMPT=*prompt-text*
Specifies a prompt to be used instead of the default prompt.

Remarks

If the **PROMPT=** attributes is not used, the element displays the following message followed by a text box: "You can search this index. Type the keyword(s) you want to search for:" When the user enters text and presses ENTER, that text is posted back to the page's URL as a query.

Example

```
<ISINDEX ACTION="http://intranet/search" PROMPT="Type in keywords here.">
```

KBD

Text to be entered at the keyboard. Renders text in fixed-width and boldface type.

<KBD>

Example

```
<KBD>This user should enter this text.</KBD>
```

Source

HTML 2

LI

Denotes one item of a list. In a DIR, MENU, OL, or UL block, denotes a new list item.

<LI
TYPE=*order-type*
VALUE=*n*>

Parameters

TYPE=*order-type*
Changes the style of an ordered list. The *order-type* can be one of these values:

A	Use large letters
a	Use small letters
I	Use large Roman numerals
i	Use small Roman numerals
1	Use numbers.

VALUE=*n*
Changes the count of ordered lists as they progress.

Example

```
<DIR> <LI>Art
<LI>History
<LI>Literature
<LI>Sports
<LI>Entertainment
<LI>Science</DIR>
```

LINK

The LINK element establishs a hierarchical organization for in navigating between documents. The LINK element must reside within the HEAD element. The HEAD element may contain several LINK elements.

<**LINK**
HREF=*URL*>

Parameters

HREF=*URL*
Specifies the URL which has a relationship to the current document.

Example

```
<LINK HREF="http://www.microsoft.com/newdocnewdoc.htm">
```

LISTING

Renders text in fixed-width type.

<LISTING>

Example

```
<LISTING>Here's some plain text.</LISTING>
```

MAP

Specifies a collection of hot spots for a client-side image map.

<MAP
NAME=*name*>

Parameters

NAME=*name*
Gives the MAP a name so it can be referred to later. See below for an example of a client-side image map.

Example

```
<MAP NAME="map1">
     <AREA ... >
    <AREA ... >
</MAP>
```

MARQUEE

Creates a scrolling text marquee.

<**MARQUEE**
ALIGN=*align-type*
BEHAVIOR=*type*
BGCOLOR=*color*
DIRECTION=*direction*
HEIGHT=*n*
HSPACE=*n*
LOOP=*n*
SCROLLAMOUNT=*n*
SCROLLDELAY=*n*
VSPACE=*n*
WIDTH=*n*>

Parameters

ALIGN=*align-type*

Specifies how the surrounding text should align with the marquee. The *align-type* can be one of these values:

TOP	Surrounding text aligns with the top of the marquee.
MIDDLE	Surrounding text aligns with the middle of the marquee.
BOTTOM	Surrounding text aligns with the bottom of the marquee.

BEHAVIOR=*type*

Specifies how the text should behave. The *type* can be one of these values:

SCROLL	Start completely off one side, scroll all the way across and completely off, and then start again. This is the default.
SLIDE	Start completely off one side, scroll in, and stop as soon as the text touches the other margin.
ALTERNATE	Bounce back and forth within the marquee.

BGCOLOR=*color*

Specifies a background color for the marquee. The *color* can be either a hexadecimal number (optionally preceded by a number (#) sign) specifying a red, green, and blue color value, or a predefined color name as described in Color.

DIRECTION=*direction*

Specifies which direction the text should scroll. The *direction* can be LEFT or RIGHT. The default is LEFT, which means scrolling to the left from the right.

HEIGHT=*n*

Specifies the height of the marquee, either in pixels or as a percentage of the screen height. To specify a percentage, the *n* must be end with a percent (%) sign.

HSPACE=*n*

Specifies left and right margins for the outside of the marquee, in pixels.

LOOP=*n*

Specifies how many times a marquee will loop when activated. If *n*=-1, or if LOOP=INFINITE is specified, it will loop indefinitely.

SCROLLAMOUNT=*n*

Specifies the number of pixels between each successive draw of the marquee text.

SCROLLDELAY=*n*

Specifies the number of milliseconds between each successive draw of the marquee text.

VSPACE=*n*

Specifies top and bottom margins for the outside of the marquee, in pixels.

WIDTH=*n*

Sets the width of the marquee, either in pixels or as a percentage of the screen width. To specify a percentage, the *n* must end with a percent (%) sign.

Example

```
<MARQUEE DIRECTION=RIGHT BEHAVIOR=SCROLL SCROLLAMOUNT=10
SCROLLDELAY=200>This is a scrolling marquee.</MARQUEE>
```

MENU

Denotes a list of items. Specifies that the following block consists of individual items, each beginning with an LI element.

<MENU>

Example

```
<MENU>
<LI>This is the first item in the menu.
<LI>And this is the second item in the menu.
</MENU>
```

META

Provides information about an HTML document to browsers, search engines and other applications.

<META
HTTP-EQUIV=*response*
CONTENT=*description*
NAME=*description*
URL=*url*
>

Parameters

CONTENT=
Defines the meta-information content to be associated with the given name or HTTP response header. Can be used with URL= and a date and time specification to reload a document at a specified interval. See the Author's Guide section "Client Pull" or the examples that follow.

HTTP-EQUIV=
Binds the element to an HTTP response header. This information is then used based on the application reading the header. See the examples that follow.

NAME=
A description of the document.

URL=
The document's URL.

Examples If the document contains:

```
<META HTTP-EQUIV="Expires"
      CONTENT="Tue, 04 Dec 1996 21:29:02 GMT">
<meta http-equiv="Keywords" CONTENT="HTML, Reference">
<META HTTP-EQUIV="Reply-to"
      content="anybody@microsoft.com">
<Meta Http-equiv="Keywords" CONTENT="HTML Reference Guide">

then the server may include the following header fields:

Expires: Tue, 04 Dec 1996 21:29:02 GMT
Keywords: HTML, Reference
Reply-to: anybody@microsoft.com

as part of the HTTP response to a `GET' or `HEAD' request for that
document.

<ex><HTML>
<HEAD>
<META HTTP-EQUIV="REFRESH" CONTENT=2>
<TITLE>Reload Document</TITLE>
</HEAD>
<BODY>
<P>This document will be reloaded every two seconds.
</BODY>
```

```
</HTML>

<HTML>
<HEAD>
<META HTTP-EQUIV="REFRESH" CONTENT="5; URL=http://www.sample.com/next.htm">
<TITLE>Load Next Document</TITLE>
</HEAD>
<BODY>
<P>After five seconds have elapsed, the document
"http://www.sample.com/next.htm" will be loaded.
</BODY>
</HTML>
```

NOBR

Turns off line breaking. Renders text without line breaks.

<NOBR>

Example

```
<NOBR>Here's a line of text I don't want to be broken . . . here's the end
of the line.</NOBR>
```

NOFRAMES

Content viewable only by browsers that do not support frames. Browsers that support frames will not display content between the beginning and ending **NOFRAMES** tags. You can create a page that is compatible with both types of browser by using NOFRAMES.

<NOFRAMES>

Example

```
<FRAMESET>
 <NOFRAMES>You need Internet Explorer 3.0 to view frames!</NOFRAMES>
</FRAMESET>
```

OBJECT

Inserts an object, such as an image, document, applet or control, into the HTML document.

<**OBJECT**
ALIGN=*align-type*
BORDER=*n*
CLASSID=*url*
CODEBASE=*url*
CODETYPE=*codetype*
DATA=*url*
DECLARE
HEIGHT=*n*
HSPACE=*n*
NAME=*url*
SHAPES
STANDBY=*message*
TYPE=*type*
USEMAP=*url*
VSPACE=*n*
WIDTH=*n*>

Parameters

ALIGN=*align-type*

Sets the alignment for the object. The *align-type* can be one of these values:

BASELINE	The bottom of the object aligns with the baseline of surrounding text.
CENTER	The object is centered between left and right margins. Subsequent text starts on the next line after the object.
LEFT	The object aligns with the left margin, and subsequent text wraps along the right side of the object.
MIDDLE	The middle of the object aligns with the baseline of surrounding text.
RIGHT	The object aligns with the right margin, and subsequent text wraps along the left side of the object.
TEXTBOTTOM	The bottom of the object aligns with the bottom of surrounding text.
TEXTMIDDLE	The middle of the object aligns with the midpoint between the baseline and the x-height of the surrounding text.
TEXTTOP	The top of the object aligns with the top of surrounding text.

BORDER=*n*

Specifies the width of the border if the object is defined to be a hyperlink.

CLASSID=*url*

Identifies the object implementation. The syntax of the *url* depends on the object type. For example, for registered ActiveX controls, the syntax is: **CLSID:***class-identifier*.

CODEBASE=*url*

Identifies the codebase for the object. The syntax of the *url* depends on the object.

CODETYPE=*codetype*

Specifies the Internet media type for code.

DATA=*url*
Identifies data for the object. The syntax of the *url* depends on the object.

DECLARE
Declares the object without instantiating it. Use this when creating cross-references to the object later in the document or when using the object as a parameter in another object.

HEIGHT=*n*
Specifies the suggested height for the object.

HSPACE=*n*
Specifies the horizontal gutter. This is the extra, empty space between the object and any text or images to the left or right of the object.

NAME=*url*
Sets the name of the object when submitted as part of a form.

SHAPES
Specifies that the object has shaped hyperlinks.

STANDBY=*message*
Sets the message to show while loading the object.

TYPE=*type*
Specifies the Internet media type for data.

USEMAP=*url*
Specifies the image map to use with the object.

VSPACE=*n*
Specifies the vertical gutter. This is the extra, empty space between the object and any text or images above or below the object.

WIDTH=*n*
Specifies the suggested width for the object.

Remarks

The end-tag is required.

An object can contain any elements ordinarily used within the body of an HTML document, including section headings, paragraphs, lists, forms and even nested objects.

OL

Draws lines of text as an ordered list. Specifies that the following block consists of individual items, each beginning with an LI tag. The items are numbered.

<OL
START=*n*
TYPE=*order-type* **>**

Parameters

START=*n*
Specifies a starting number for the list.

TYPE=*order-type*
Changes the style of the list. The *order-type* can be one of these values:

A	Use large letters
a	Use small letters
I	Use large Roman numerals
i	Use small Roman numerals
1	Use numbers.

Example

```
<OL>
<LI>This is the first item in the list.
<LI>And this is the second item in the list.
</OL>

<ex><OL START=3>
<LI>This is item number 3.
</OL>

<OL TYPE=A>
<LI>This is item A.
</OL>
```

OPTION

Denotes one choice in a list box. In a SELECT block, denotes one of the choices that will appear in the list.

<**SELECT**
SELECTED
VALUE=*value*>

Parameters

SELECTED
Indicates that this item is the default. If not present, item #1 becomes the default.

VALUE=*value*
Indicates the value that will be returned if this item is chosen.

P

Denotes a paragraph. Inserts a paragraph break and denotes a paragraph.

<P
ALIGN=*align-type***>**

Parameters

ALIGN=*align-type*
Sets the alignment of the paragraph. The *align-type* can be CENTER, LEFT, or RIGHT. Default is left alignment.

Remarks

The end-tag is optional.

Example

```
<P>This is a paragraph.</P>
```

PARAM

Sets property values for a given object.

<**PARAM**
NAME=*name*
VALUE=*value*
VALUETYPE=*type*
TYPE=*type*>

Parameters

NAME=*name*
Specifies the property name.

VALUE=*value*
Specifies the property value. The value is passed to the object without change except that any character or numeric character entities are replaced with their corresponding character values.

VALUETYPE=*type*
Specifies how to interpret the value. The *type* can be one of these values:

DATA	The value is data. This is the default value type.
REF	The value is a URL.
OBJECT	The value is a URL of an object in the same document.

TYPE=*type*
Specifies the Internet media type.

Remarks

This element is valid only within an OBJECT element. The end-tag is optional.

PLAINTEXT

Renders text in fixed-width type without processing tags. Renders text in fixed-width type. Also turns off HTML parsing until the browser encounters the </PLAINTEXT> tag.

<PLAINTEXT>

Example

```
<PLAINTEXT> Here's a sample of HTML: <A HREF="sample.url">This is a
shortcut to sample.</A></PLAINTEXT>
```

PRE

Renders text in fixed-width type.

<PRE>

Example

```
<PRE>Here's some plain text.</PRE>
```

Source

HTML 2

S

Renders text in strikethrough type.

<S>

Example

```
<S>This text has a line through it.</S>
```

SAMP

Specifies sample text. Renders text in a small font. (If no FONT FACE is specified, the font used is fixed-width.)

<SAMP>

Example

```
<SAMP>Here is some text in a small fixed-width font.</SAMP>
```

SELECT

Denotes a list box or dropdown list.

<SELECT
MULTIPLE
NAME=*name*
SIZE=*n*>

Parameters

MULTIPLE
Indicates that multiple items can be selected.

NAME=*name*
Specifies a name for the list.

SIZE=*n*
Specifies the height of the list control.

Example

```
<SELECT NAME="Cars" MULTIPLE SIZE="1">
    <OPTION VALUE="1">BMW
    <OPTION VALUE="2">PORSCHE
    <OPTION VALUE="3" SELECTED>MERCEDES
</SELECT>
```

SMALL

Makes text one size smaller.

<SMALL>

Example

```
<SMALL>This text is smaller</SMALL>
```

SPAN

Use to apply style information to text within a document. SPAN can be used to do localized formatting to text using STYLE as an attribute.

<SPAN>
STYLE=

Example

```
<SPAN STYLE="margin-left: 1.0in"> This paragraph is 1.0 inches from the
left margin.<SPAN>
```

STRIKE

Renders text in strikethrough type.

<STRIKE>

Example

```
<STRIKE>This text has a line through it.</STRIKE>
```

STRONG

Emphasizes the text. Usually displays the text in bold.

<STRONG>

Example

```
<STRONG>This text will be bold.</STRONG>
```

SUB

Renders text in subscript.

<SUB>

Example

```
<SUB>This text is rendered as subscript.</SUB>
```

SUP

Renders text in superscript.

<SUP>

Example

```
<SUP>This text is rendered as superscript.</SUP>
```

TABLE

Creates a table. The table is empty unless you create rows and cells using the TR, TD, and TH elements.

<**TABLE**
ALIGN=*align-type*
BACKGROUND=*url*
BGCOLOR=*color*
BORDER=*n*
BORDERCOLOR=*color*
BORDERCOLORDARK=*color*
BORDERCOLORLIGHT=*color*
CELLPADDING=*n*
CELLSPACING=*n*
COLS=*n*
FRAME=*frame-type*
RULES=*rules*
WIDTH=*n*

Parameters

ALIGN=*align-type*
Specifies the table alignment. The *align-type* can be one of these values:

LEFT	The table is left-aligned. This is the default alignment.
RIGHT	The table is right-aligned. If the table is less than the width of the window, text following the table wraps along the left side of the table.

BACKGROUND=*url*
Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*
Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDER=*n*
Size in pixels of the table border. The default is zero.

BORDERCOLOR=*color*
Sets border color and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORLIGHT=*color*
Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORDARK, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORDARK=*color*
Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORLIGHT, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

CELLPADDING=*n*
Amount of space, in pixels, between the sides of a cell and its contents.

CELLSPACING=*n*
Amount of space, in pixels, between the frame (exterior) of the table and the cells in the table.

COLS=*n*
Number of columns in the table. If given, this attribute may speed up processing of tables, especially lengthy tables.

WIDTH=*n*
Sets the width of the table in pixel or as a percentage of the window. To set a percentage, the *n* must end with a percent (%) sign.

FRAME=*frame-type*
Specifies which sides of a frame (outer borders) are displayed. The *frame-type* can be one of these values:

VOID	Removes all outside table borders.
ABOVE	Displays a border on the top side of the table frame.
BELOW	Displays a border on the bottom side of the table frame.
HSIDES	Displays a border on the top and bottom sides of the table frame.
LHS	Displays a border on the left-hand side of the table frame.
RHS	Displays a border on the right-hand side of the table frame.
VSIDES	Displays a border on the left and right sides of the table frame.
BOX	Displays a border on all sides of the table frame.
BORDER	Displays a border on all sides of the table frame.

RULES=*rule-type*
Specifies which dividing lines (inner borders) are displayed. The *rule-type* can be one of these values:

NONE	Removes all interior table borders.
GROUPS	Displays horizontal borders between all table groups. Groups are specified by the THEAD, TBODY, TFOOT, and COLGROUP elements.
ROWS	Displays horizontal borders between all table rows.
COLS	Displays vertical borders between all table columns.
ALL	Displays a border on all rows and columns.

Remarks

The end-tag is required.

The optional THEAD, TBODY, TFOOT, COLGROUP, and COL elements can be used to organize a table and apply attributes across columns and groups of columns.

Example

```
<TABLE BORDER=1 WIDTH=80%>
<THEAD>
<TR>
    <TH>Heading 1</TH>
    <TH>Heading 2</TH>
</TR>
<TBODY>
<TR>
    <TD>Row 1, Column 1 text.</TD>
    <TD>Row 1, Column 2 text.</TD>
```

```
</TR>
<TR>
    <TD>Row 2, Column 1 text.</TD>
    <TD>Row 2, Column 2 text.</TD>
</TR>
</TABLE>
```

TBODY

Defines the table body. Use this element to distinguish the rows in the table header or footer from those in the main body of the table.

<TBODY>

Remarks

If a table does not have a header or footer (does not have a THEAD or TFOOT element), the **TBODY** element is optional. The end-tag is always optional.

You can use the **TBODY** element more than once in a table. This is useful for dividing lengthy tables into smaller units and for controlling the placement of horizontal rules.

Example

```
<TABLE>
<THEAD>
<TR>
    ...
</TR>
<TBODY>
<TR>
    ...
</TR>
</TBODY>
</TABLE>
```

TD

Creates a cell in a table.

<**TD**
ALIGN=*align-type*
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*
COLSPAN=*n*
NOWRAP=NOWRAP
ROWSPAN=*n*
VALIGN=*align-type*>

Parameters

ALIGN=*align-type*

Specifies the horizontal alignment of text in the cell. The *align-type* can be one of these values:

LEFT	The table is left-aligned.
CENTER	Text is centered.
RIGHT	The table is right-aligned.

By default, text is centered.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*

Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLOR=*color*

Sets border color and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORDARK, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORLIGHT, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

VALIGN=*align-type*

Specifies that the vertical alignment of text in the cell. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

Remarks

This element is valid only within a row in a table, that is, you must use a TR element before using **TD**. All attributes are optional. The end-tag is optional.

TEXTAREA

Creates a multiple-line text entry control in which the user can enter and edit text.

<**TEXTAREA**
COLS=*n*
NAME=*name*
ROWS=*n*>

Parameters

COLS=*n*
Sets the width, in characters, of the text area.

NAME=*n*
Sets the name of the text area. This name is used when the element is used within a FORM element.

ROWS=*n*
Sets the height, in characters, of the text area.

Remarks

The end-tag is required. Any text between the start and end-tags is used as the initial value for the control.

TFOOT

Defines the table footer. Use this element to distinguish the rows in the table footer from those in the header or main body of the table.

<TFOOT>

Remarks

The table footer is optional, but if given only one footer is allowed. The **TFOOT** element is valid only within a table; you must use a TABLE element before using this element. The end-tag is optional.

Example

```
<TABLE>
<TBODY>
    <TR>
    ...
    </TR>
<TFOOT>
    <TR>
    ...
    </TR>
</TABLE>
```

TH

Creates a row or column heading in a table. The element is similar to the TD element but emphasizes the text in the cell to distinguish it from text in TD cells.

<**TH**
ALIGN=*align-type*
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*
COLSPAN=*n*
NOWRAP=NOWRAP
ROWSPAN=*n*
VALIGN=*align-type*>

Parameters

ALIGN=*align-type*

Specifies the alignment of text in the cell. The *align-type* can be one of these values:

LEFT	Text is left-aligned.
CENTER	Text is centered.
RIGHT	Text is right-aligned.

By default, text is centered.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*

Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLOR=*color*

Sets border color and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORDARK, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORLIGHT, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

COLSPAN=*n*

Number of table columns this cell spans.

NOWRAP=NOWRAP

Prevents word wrapping with the cell. Lines of text appear as given in the HTML document.

ROWSPAN=*n*

Number of table rows this cell spans.

VALIGN=*align-type*

Specifies that the vertical alignment of text in the table. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

Remarks

This element is valid only within a row in a table, that is, you must use a TR element before using **TH**. All attributes are optional. The end-tag is optional.

THEAD

Defines the table header. Use this element to distinguish the rows in the table header from those in the footer or main body of the table.

<THEAD>

Remarks

The table header is optional, but if given only one header is allowed. The **THEAD** element is valid only within a table; you must use a TABLE element before using this element. The end-tag is optional.

Example

```
<TABLE>
<THEAD>
    <TR>
    ...
    </TR>
<TBODY>
    <TR>
    ...
    </TR>
</TABLE>
```

TITLE

Specifies a title for the document. Internet Explorer uses this for the window caption.

This element is valid only within the HEAD element. The end-tag is required.

<TITLE>

Example

```
<HEAD>
<TITLE>"Welcome To Internet Explorer!"</TITLE>
</HEAD>
```

TR

Creates a row in a table.

<**TR**
ALIGN=*align-type*
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*
VALIGN=*align-type*>

Parameters

ALIGN=*align-type*

Specifies the alignment of text in the cells in the row. The *align-type* can be one of these values:

LEFT	Text is left-aligned.
CENTER	Text is centered.
RIGHT	Text is right-aligned.

By default, text is centered.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*

Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLOR=*color*

Sets border color and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORDARK, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORLIGHT, and must be used with the BORDER attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

VALIGN=*align-type*

Specifies that the vertical alignment of text in the cells of the row. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

TT

Teletype. Renders text in fixed-width type.

<TT>

Example

```
<TT>Here's some plain text.</TT>
```

U

Renders text underlined.

<U>

Example

```
<U>This text has a line under it.</U>
```

UL

Draws lines of text as a bulleted list. Specifies that the following block consists of individual items, each beginning with an LI tag. The items are bulleted.

<UL>

Example

```
<UL>
    <LI>This is the first bulleted item in the list.
    <LI>And this is the second bulleted item in the list.
</UL>
```

VAR

Placeholder text for a variable. Displays text in a small fixed-width type.

<VAR>

Example

```
Enter the <VAR>filename</VAR> in the dialog box.
```

WBR

Inserts a soft linebreak in a block of NOBR text.

Example

```
<NOBR> This line of text will not break, no matter how narrow the window
gets.<WBR> This one, however<WBR>, will.</NOBR>
```

XMP

Example text. Displays text in fixed-width type.

<XMP>

Example

```
<XMP>Here's some plain text.</XMP>
```

Introduction

This document describes WebBrowser, an ActiveX™ control that developers can use to add Internet browsing capabilities to their applications, and WebBrowserApp, an OLE Automation object that developers can use to control the Microsoft® Internet Explorer (IE) application from within an application.

Using the WebBrowser Control

The WebBrowser control adds browsing, document viewing, and data downloading capabilities to your applications. It allows the user to browse sites on the Internet's World Wide Web, as well as folders in the local file system and on a network. The WebBrowser control supports Web browsing through both point-and-click hyperlinking and Uniform Resource Locator (URL) navigation. Also, the control maintains a history list that allows the user to browse forward and backward through previously browsed sites, folders, and documents.

The WebBrowser control includes support for parsing and displaying HTML-encoded documents. It is also an ActiveX document container that can host any ActiveX document. This means that richly formatted documents, such as a Microsoft Excel spreadsheet or Microsoft Word documents, can be opened and edited in-place from within the WebBrowser control. WebBrowser is also an ActiveX control container that can host any ActiveX control.

Possible Uses

Some possible uses of the WebBrowser control are:

- To enable the user to navigate to and view HTML documents on the local computer, network, or World Wide Web.
- To provide a single frame in which the user can view and edit all types of ActiveX documents.
- To create a customized Web browsing application based on the WebBrowser control.

Navigate to a New Location with the Navigate Method

The WebBrowser control can browse to any location in the local file system, on the network, or on the World Wide Web. You use the Navigate method to tell the control which location to browse to. The first parameter is a string that contains the name of the location. To browse to a location in the local file system or on the network, specify the full path to the file system location or the Universal Naming Convention (UNC) name of the location on the network. To browse to a site on the World Wide Web, specify the URL of the site. By including a text box in your application, you can let the user specify the location to browse to, and then pass the location to the Navigate method.

The Navigate method allows you to target a specific frame on an HTML page, causing the WebBrowser control to display a Web site or file system location in that frame. First, you would call the Navigate method and specify the URL of an HTML page that contains a frame. Then, by specifying the name of the frame in subsequent calls to Navigate, you can direct the control to display subsequent locations within that frame.

You can use the LocationName and LocationURL properties to retrieve information about the location that the WebBrowser control is currently displaying. If the location is an HTML page on the World Wide Web, LocationName retrieves the title of that page, and LocationURL retrieves the URL of that page. If the location is a folder or file on the network or local computer, LocationName and LocationURL both retrieve the UNC or full path of the folder or file.

An HTML author can create a hyperlink that causes an Internet server to execute a program whenever the hyperlink is selected. The hyperlink typically passes data to the program during the HTTP transaction, and the program processes the data. The Navigate method supports navigating to such a hyperlink, and includes parameters that allow you to specify data to pass to the program. You can specify an HTTP header, HTTP post data, and the URL of the referring document. The HTTP header specifies such things as the action required of the server, the type of data being passed to the server, or a status code. The HTTP post data is the actual data for the program to process. The referring document is the HTML document that contains the hyperlink.

Update Cached Pages with the Refresh or Refresh2 Method

The WebBrowser control stores Web pages from recently visited sites in cached memory on the user's hard disk so that, when the control revisits a site, it can save time by reloading the page from the cache rather than downloading it again from the Internet server. You can force the WebBrowser control to re-download a page by using the Refresh or Refresh2 method—this ensures that the user is viewing the current version of the page. Also, you can prevent the control from using the cache by specifying the navNoReadFromCache and navNoWriteToCache flags when calling the Navigate method.

Navigate the History List with the GoBack and GoForward Methods

During a browsing session, the WebBrowser control maintains a history list of all Web sites that it visited during the session (unless you specify navNoHistory when calling the Navigate method). You can direct the control to browse backward and forward through the sites in the list by using the GoBack and GoForward methods.

You can also use the GoHome method to cause the WebBrowser control to return to the user's home or start page, and the GoSearch method to go to the user's Web searching page.

Control Browsing Operations with the Busy Property and Stop Method

You can use the Busy property to determine if the WebBrowser control is in the process of navigating to a new location or downloading a file. If the control is busy, you can use the Stop method to cancel the navigation or the download before it is finished.

Hide or Show the WebBrowser Control with the Visible Property

By default, the WebBrowser control is hidden when it is first created, but it becomes visible after calling the Navigate or GoSearch method. You can also set the Visible property to **True** to show the control, or to **False** to hide it.

WebBrowser Events

The WebBrowser control fires a number of different events to notify an application of user- and browser-generated activity. When the control is about to navigate to a new location, it fires a BeforeNavigate event that specifies the URL or path of the new location and any other data that will be transmitted to the Internet server through the HTTP transaction. The data can include the HTTP header, HTTP post data, and the URL of the referrer. The BeforeNavigate event also includes a cancel flag that you can set to **False** to cancel the navigation. The WebBrowser control fires the NavigateComplete event after it has navigated to a new location. This event includes the same information as BeforeNavigate, except NavigateComplete does not include the cancel flag.

Whenever the WebBrowser control is about to begin a download operation, it fires the DownloadBegin event. The control fires a number of ProgressChange events as the operation progresses, and then fires the DownloadComplete event after completing the operation. Applications typically use these three events to indicate the progress of the download operation, often by displaying a progress bar. An application would show the progress bar in response to DownloadBegin, update the progress bar in response to ProgressChange, and hide it in response to DownloadComplete.

When an application calls the Navigate method with the navOpenInNewWindow flag, the WebBrowser control fires the NewWindow event before navigating to the new location. The event includes information about the new location, and a flag that indicates whether the application or the control is to create the new window. Set this flag to **True** if your application will create the window, or to **False** if the WebBrowser control should create it.

Using the WebBrowserApp Object

The WebBrowserApp object allows an application to create and manipulate an instance of Internet Explorer through OLE Automation. The WebBrowserApp object supports the same properties and methods as the WebBrowser control, plus several that the WebBrowser control does not support. This section describes the additional properties and methods supported by the WebBrowserApp object. For information about the properties and methods supported by both objects, see Using the WebBrowser Control.

Show and Hide User Interface Elements

The WebBrowserApp object supports a number of properties that you can use to show and hide user-interface elements of Internet Explorer, including the menu bar, the status bar, and the toolbar. The MenuBar, StatusBar, and ToolBar properties all take Boolean values. If the value of any of these properties is **True**, the corresponding user interface element is shown; if the value is **False**, the corresponding element is hidden. You can set or retrieve the text of the status bar by using the StatusText property.

The FullScreen property determines whether Internet Explorer is in full-screen or normal window mode. In full-screen mode, the Internet Explorer main window is maximized and the status bar, toolbar, menu bar, and title bar are hidden.

Retrieve Information About Internet Explorer

You can use the Name property to retrieve the name of the Internet Explorer application, or the Path property to retrieve the application's full path to the Internet Explorer application. The FullName property is similar to Path, except it retrieves the full path, including the file name, of the executable file that contains the Internet Explorer application. The HWND property retrieves the window handle of the Internet Explorer main window.

Use the Quit method to close the Internet Explorer application.

WebBrowser Object

The **WebBrowser** object is an ActiveX control that allows you to add browsing capabilities to your applications. The WebBrowser control can be used to browse sites on the World Wide Web, as well as directories on the local computer and on network servers.

Properties

Application, Busy, Container, Document, Height, Left, LocationName, LocationURL, Parent, Top, TopLevelContainer, Type, Width

Methods

GoBack, GoForward, GoHome, GoSearch, Navigate, Refresh, Refresh2, Stop

Events

BeforeNavigate, CommandStateChange, DownloadBegin, DownloadComplete, NavigateComplete, NewWindow, ProgressChange, StatusTextChange, TitleChange

WebBrowserApp Object

The WebBrowserApp object allows an application to create and control an instance of the Microsoft Internet Explorer application.

Properties

Application, Busy, Container, Document, FullName, FullScreen, Height, HWND, Left, LocationName, LocationURL, MenuBar, Name, Parent, Path, StatusBar, StatusText, ToolBar, Top, TopLevelContainer, Type, Visible, Width

Methods

ClientToWindow, GetProperty, GoBack, GoForward, GoHome, GoSearch, Navigate, PutProperty, Quit, Refresh, Refresh2, Stop

Events

BeforeNavigate, CommandStateChange, DownloadBegin, DownloadComplete, NavigateComplete, NewWindow, ProgressChange, PropertyChange, Quit, StatusTextChange, TitleChange, WindowActivate, WindowMove, WindowResize

Application Property

Returns the automation object supported by the application that contains the WebBrowser control if the object is accessible; otherwise, this property returns the WebBrowser control's automation object.

object.**Application**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

Busy Property

Returns a Boolean value specifying whether the WebBrowser control or Internet Explorer is engaged in a navigation or downloading operation.

object.**Busy**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Return Values

The **Busy** property returns these values:

Value	Description
True	A download or other operation is in progress.
False	No download or other operation is in progress.

Applies To

WebBrowser, WebBrowserApp

Container Property

Returns an object that evaluates to the container of the WebBrowser control, if any.

object.**Container**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

Document

Document Property

Returns the automation object of the active document, if any.

object.**Document**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

Container

FullName Property

Returns a string that evaluates to the fully qualified path of the executable file that contains the Internet Explorer application.

object.**FullName**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

FullScreen Property

Returns or sets a value indicating whether Internet Explorer is in full-screen or normal window mode. In full-screen mode, the Internet Explorer main window is maximized and the status bar, toolbar, menu bar, and title bar are hidden.

object.**FullScreen** [**=** *value*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

value
Optional. A Boolean expression that determines whether Internet Explorer is in full-screen or normal window mode. If **True**, the object is in full-screen mode; if **False**, it is in normal mode.

Applies To

WebBrowserApp

Height Property

Returns or sets the vertical dimension, in pixels, of the frame window that contains the WebBrowser control.

object.**Height** [**=** *height*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

height
Optional. A long integer value specifying the vertical dimension of the frame window, in pixels.

Applies To

WebBrowser, WebBrowserApp

See Also

Width

HWND Property

Returns the handle of the Internet Explorer main window.

object.**HWND**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

Left Property

Returns or sets the distance between the internal left edge of the WebBrowser control and the left edge of its container.

object.**Left** [**=** *distance*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

distance
Optional. A long integer expression specifying the distance between the internal left edge of the WebBrowser control and the left edge of its container.

Remarks

The **Left** property is measured in units depending on the coordinate system of its container. The values for this property change as the object is moved by the user or by code.

Applies To

WebBrowser, WebBrowserApp

LocationName Property

Returns a string that contains the name of the resource that the WebBrowser control is currently displaying. If the resource is an HTML page on the World Wide Web, the name is the title of that page. If the resource is a folder or file on the network or local computer, the name is the UNC or full path of the folder or file.

object.**LocationName**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

LocationURL

LocationURL Property

Returns a string that contains the URL of the resource that the WebBrowser control or Internet Explorer is currently displaying. If the resource is a folder or file on the network or local computer, the name is the UNC or full path of the folder or file.

object.**LocationURL**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

LocationName

MenuBar Property

Returns or sets a value that determines whether the Internet Explorer menu bar is visible or hidden.

object.**MenuBar** [**=** *value*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

value
Optional. A Boolean expression that determines whether the menu bar is visible. If **True**, the menu bar is visible; if **False**, it is hidden.

Applies To

WebBrowserApp

Name Property

Returns a string that evaluates to the name of the Internet Explorer application; that is, "Microsoft Internet Explorer."

object.**Name**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

Parent Property

Returns the form on which the WebBrowser control is located, or the automation object supported by Internet Explorer.

object.**Parent**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

Path Property

Returns a string that evaluates to the full path of the Internet Explorer application.

object.**Path**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

StatusBar Property

Returns or sets a value that determines whether the status bar is visible.

object.**StatusBar** [**=** *value*]

Parts

object
Required. An object expression that evaluates to a WebBrowserApp object.

value
Optional. A Boolean expression that determines whether the status bar is visible. If **True**, the status bar is visible; if **False**, it is hidden.

Applies To

WebBrowserApp

See Also

StatusText, StatusTextChange, ToolBar, Visible

StatusText Property

Returns or sets the text for the status bar.

object.**StatusText** [**=** *value*]

Parts

object
Required. An object expression that evaluates to a WebBrowserApp object.

value
Optional. A string that evaluates to the text for the status bar.

Applies To

WebBrowserApp

See Also

StatusBar, StatusTextChange

ToolBar Property

Returns or sets a value that determines whether the toolbar is visible.

object.**ToolBar** [**=** *value*]

Parts

object
Required. An object expression that evaluates to a WebBrowserApp object.

value
Optional. A Boolean expression that determines whether the toolbar is visible. If **True**, the toolbar is visible; if **False**, it is hidden.

Applies To

WebBrowserApp

See Also

StatusBar, Visible

Top Property

Returns or sets the distance between the internal top edge of the WebBrowser control and the top edge of its container.

object.**Top** [**=** *value*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

value
Optional. A long integer expression specifying distance.

Remarks

The **Top** property is measured in units depending on the coordinate system of its container. The values for this property change as the object is moved by the user or by code.

Applies To

WebBrowser, WebBrowserApp

TopLevelContainer Property

Returns a Boolean value indicating whether the given object is a top-level container.

object.**TopLevelContainer**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

Type Property

Returns a string expression that specifies the type name of the contained document object.

object.**Type**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

Visible Property

Returns or sets a value indicating whether Internet Explorer is visible or hidden.

object.**Visible** [**=** *value*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

value
Optional. A Boolean expression specifying the visible state of Internet Explorer. If **True**, the window is visible; if **False**, it is hidden.

Applies To

WebBrowserApp

Width Property

Returns or sets the horizontal dimension, in pixels, of the frame window that contains the WebBrowser control.

object.**Width** [**=** *width*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

width
Optional. A long integer value specifying the horizontal dimension of the frame window, in pixels.

Applies To

WebBrowser, WebBrowserApp

See Also

Height

ClientToWindow Method

Converts the client coordinates of a point to window coordinates. Client coordinates are relative to the upper-left corner of the client area; window coordinates are relative to the upper-left corner of a window.

object.**ClientToWindow** *pcx*, *pcy*

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

pcx
Required. A long integer value that specifies the x-coordinate of the point in client coordinates. When **ClientToWindow** returns, this variable contains the x-coordinate of the point in window coordinates.

pcy
Required. A long integer value that specifies the y-coordinate of the point in client coordinates. When **ClientToWindow** returns, this variable contains the y-coordinate of the point in window coordinates.

Applies To

WebBrowserApp

GetProperty Method

Retrieves the current value of a property associated with the given object.

object.**GetProperty** *szProperty*, *vtValue*

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

szProperty
Required. A string expression that contains the name of the property to retrieve.

vtValue
Required. A variable that receives the current value of the property.

Applies To

WebBrowserApp

See Also

PropertyChange, PutProperty

GoBack Method

Navigates backward one item in the history list.

object.**GoBack**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

GoForward, GoHome, GoSearch

GoForward Method

Navigates forward one item in the history list.

object.**GoForward**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

GoBack, GoHome, GoSearch

GoHome Method

Navigates to the current home or start page, as specified in the Internet Explorer Options dialog box and Internet Control Panel.

object.**GoHome**

Parts

object
Required. An object expression that evaluates to object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

GoBack, GoForward, GoSearch

GoSearch Method

Navigates to the current search page, as specified in the Internet Explorer Options dialog box and Internet Control Panel.

object.**GoSearch**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

GoBack, GoForward, GoHome

Navigate Method

Navigates to the resource identified by a Universal Resource Locator (URL), or to the file identified by a full path.

object.**Navigate** *URL* [*Flags*,] [*TargetFrameName*,] [*PostData*,] [*Headers*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

URL
Required. A string expression that evaluates to the URL of the resource to display, or the full path of the file to display.

Flags
Optional. A constant or value that specifies whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a combination of the following values.

Constant	Value	Meaning
navOpenInNewWindow	1	Open the resource or file in a new window.
navNoHistory	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache	4	Do not read from the disk cache for this navigation.
navNoWriteToCache	8	Do not write the results of this navigation to the disk cache.

TargetFrameName
Optional. A string expression that evaluates to the name of a frame in which to display the resource.

PostData
Optional. Data to send to the server during the HTTP POST transaction. For example, the POST transaction is used to send data gathered by an HTML form. If this parameter does not specify any post data, the **Navigate** method issues an HTTP GET transaction. This parameter is ignored if *URL* is not an HTTP URL.

Headers
Optional. A value that specifies additional HTTP headers to send to the server. These headers are added to the default Internet Explorer headers. The headers can specify such things as the action required of the server, the type of data being passed to the server, or a status code. This parameter is ignored if *URL* is not an HTTP URL.

Applies To

WebBrowser, WebBrowserApp

PutProperty Method

Sets the value of a property associated with the given object.

object.**PutProperty** *szProperty*, *vtValue*

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

szProperty
Required. A string expression that contains the name of the property to set.

vtValue
Required. A variable that specifies the new value of the property.

Applies To

WebBrowserApp

See Also

GetProperty, PropertyChange

Quit Method

Closes the Internet Explorer application.

object.**Quit**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

See Also

Quit, Refresh, Refresh2, Stop

Refresh Method

Reloads the file that the WebBrowser control is currently displaying.

object.**Refresh**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

Refresh2, Stop

Refresh2 Method

Reloads the file that the WebBrowser control is currently displaying. Unlike the Refresh method, this method contains a parameter that specifies the refresh level.

object.**Refresh2** [*Level*]

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Level
Optional. A constant or value that specifies the refresh level. It can be one of the following constants or values.

Constant	Value	Meaning
refreshAll	0	Perform a full refresh that includes sending a "pragma:nocache" header to the server (HTTP URLs only).
refreshDontSendNoCache	1	Perform a lightweight refresh that does not include sending the HTTP "pragma:nocache" header to the server.

Remarks

The "pragma:nocache" header tells the server not to return a cached copy, but to ensure that the information is as fresh as possible. Browsers typically send this header when the user selects refresh, but the header causes problems for some servers.

Applies To

WebBrowser, WebBrowserApp

See Also

Quit, Refresh, Stop

Stop Method

Cancels any pending navigation or download operation, and stops any dynamic page elements such as background sounds and animations.

object.**Stop**

Parts

object
Required. An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

BeforeNavigate, DownloadBegin, ProgressChange, Refresh, Refresh2

BeforeNavigate Event

Occurs when the WebBrowser control is about to navigate to a different URL, which may happen as a result of external automation, internal automation from a script, or the user clicking a link or typing in the address bar. The container has an opportunity to cancel the pending navigation.

Private Sub *object*_**BeforeNavigate(ByVal** *URL* **As String, ByVal** *Flags* **As Long, ByVal** *TargetFrameName* **As String,** *PostData* **As Variant, ByVal** *Headers* **As String,** *Cancel* **As Boolean)**

Values

object
Required. An object expression that evaluates to an object in the Applies To list.

URL
A string expression that evaluates to the URL to which the browser is navigating.

Flags
Reserved for future use.

TargetFrameName
A string expression that evaluates to the name of the frame in which to display the resource, or NULL if no named frame is targeted for the resource.

PostData
Data to send to the server if the HTTP POST transaction is being used.

Headers
A value that specifies the additional HTTP headers to send to the server (HTTP URLs only). The headers can specify such things as the action required of the server, the type of data being passed to the server, or a status code.

Cancel
A Boolean value that the container can set to **True** to cancel the navigation operation, or to **False** to allow it to proceed.

Applies To

WebBrowser, WebBrowserApp

See Also

Navigate, NavigateComplete

CommandStateChange Event

Occurs when the enabled state of a command changes.

Private Sub *object*_**CommandStateChange (ByVal** *Command* **As Long, ByVal** *Enable* **As Boolean)**

Values

object
An object expression that evaluates to an object in the Applies To list.

Command
A long integer specifying the identifier of the command that changed.

Enable
A Boolean value that is **True** if the command is enabled, or **False** if not.

Applies To

WebBrowser, WebBrowserApp

DownloadBegin Event

Occurs when a navigation operation is beginning. This event is fired shortly after the BeforeNavigate event, unless the navigation is canceled. Any animation or "busy" indication that the container needs to display should be connected to this event.

Private Sub *object*_**DownloadBegin ( )**

Values

object
An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

DownloadComplete, Navigate, ProgressChange

DownloadComplete Event

Occurs when a navigation operation finished, was halted, or failed. Unlike NavigateComplete, which is fired only when a URL is successfully navigated to, this event is always fired after a navigation starts. Any animation or "busy" indication that the container needs to display should be connected to this event.

Private Sub *object*_**DownloadComplete ( )**

Values

object
An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowser, WebBrowserApp

See Also

DownloadBegin, Navigate, NavigateComplete, ProgressChange

NavigateComplete Event

Occurs after the browser has successfully navigated to a new location. The document may still be downloading (and in the case of HTML, images may still be downloading), but at least part of the document has been received from the server, and the viewer for the document has been created.

Private Sub *object*_**NavigateComplete(ByVal** *URL* **As String)**

Values

object
An object expression that evaluates to an object in the Applies To list.

URL
A string expression that evaluates to the URL that was navigated to. Note that this URL can be different from the URL that the browser was told to navigate to. One reason is that this URL is the canonicalized and qualified URL; for instance, if an application specified a URL of "www.microsoft.com" in a call to the Navigate method, the URL passed by **NavigateComplete** will be "http://www.microsoft.com/". Also, if the server has redirected the browser to a different URL, the redirected URL will be reflected here.

Applies To

WebBrowser, WebBrowserApp

See Also

BeforeNavigate, Navigate

NewWindow Event

Occurs when a new window is to be created for displaying a resource. Some actions that can cause this include the user shift-clicking on a link, the user right-clicking on a link and choosing "open in new window", or a targeted navigation to a frame name that does not yet exist. The container has an opportunity to handle the new window creation itself. If it does not, a top-level Internet Explorer window is created as a separate process.

Private *object*_**NewWindow (ByVal** *URL* **As String, ByVal** *Flags* **As Long,**
ByVal *TargetFrameName* **As String,** *PostData* **As Variant, ByVal** *Headers* **As String,**
Processed **As Boolean)**

Values

object
An object expression that evaluates to an object in the Applies To list.

URL
A string expression that evaluates to the URL of the resource being navigated to.

Flags
Reserved for future use.

TargetFrameName
A string expression that evaluates to the name of the frame in which to display the resource, or NULL if no named frame is targeted for the resource.

PostData
Data to send to the server if an HTTP POST transaction is used.

Headers
A value that specifies the HTTP headers to send to the server (HTTP URLs only). The headers can specify such things as the action required of the server, the type of data being passed to the server, or a status code.

Processed
A Boolean value that indicates whether the container intends to create the new window. Set this parameter to **True** if the container will create the window, or to **False** if a top-level Internet Explorer window is to be created.

Remarks

The preferred behavior of WebBrowser control containers is to process this event, create a new instance of the WebBrowser control, and pass all the parameters from the **NewWindow** event directly to the Navigate method on the newly created WebBrowser control. Another option for containers that cannot or do not need to create a new window is to degrade by performing the navigation in the existing window. To do this, they may process this event and then pass the parameters from this event to Navigate on the existing window.

Applies To

WebBrowser, WebBrowserApp

See Also

BeforeNavigate, Navigate, NavigateComplete

ProgressChange Event

Occurs when the progress of a download operation is updated.

Private Sub *object*_**ProgressChange(ByVal** *Progress* **As Long, ByVal** *ProgressMax* **As Long)**

Values

object
An object expression that evaluates to an object in the Applies To list.

Progress
A long integer that specifies the amount of total progress to show, or –1 when progress is complete.

ProgressMax
A long integer that specifies the maximum progress value.

Remarks

The container can use the information provided by this event to display the number of bytes downloaded so far or to update a progress indicator.

To calculate the percentage of progress to show in a progress indicator, multiply the value of *Progress* by 100 and divide by the value of *ProgressMax* (unless progress is –1, in which case the container can indicate that the operation is finished or hide the progress indicator).

Applies To

WebBrowser, WebBrowserApp

See Also

DownloadBegin, DownloadComplete, Navigate

PropertyChange Event

Occurs when the PutProperty method changes the value of a property.

Private Sub *object*_**PropertyChange(ByVal** *szProperty* **As String)**

Values

object
An object expression that evaluates to an object in the Applies To list.

szProperty
A string expression that contains the name of the property whose value has changed.

Applies To

WebBrowserApp

See Also

GetProperty, PutProperty

Quit Event

Occurs when the Internet Explorer application is ready to quit.

Private Sub *object*_**Quit(***Cancel* **As Boolean)**

Values

object
An object expression that evaluates to an object in the Applies To list.

Cancel
A Boolean value that is **True** if the last Quit was canceled, or **False** if not.

Applies To

WebBrowserApp

See Also

Quit

StatusTextChange Event

Occurs when the status bar text has changed.

Private Sub *object*_**StatusTextChange(ByVal** *Text* **As String)**

Values

object
An object expression that evaluates to an object in the Applies To list.

Text
A string containing the new status bar text.

Remarks

The container can use the information provided by this event to update the text of a status bar.

Applies To

WebBrowser, WebBrowserApp

See Also

StatusBar, StatusText

TitleChange Event

Occurs when the title of a document in the WebBrowser control becomes available or changes. For HTML, the title may change; while HTML is still downloading, the URL of the document is set as the title. After the real title (if there is one) is parsed from the HTML, the title is changed to reflect the actual title.

Private Sub *object*_**TitleChange(ByVal** *Text* **As String)**

Values

object
An object expression that evaluates to an object in the Applies To list.

Text
A string containing the new document title.

Applies To

WebBrowser, WebBrowserApp

WindowActivate Event

Occurs when the Internet Explorer main window has been activated.

Private Sub *object*_**WindowActivate( )**

Values

object
An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

See Also

WindowMove, WindowResize

WindowMove Event

Occurs when the Internet Explorer main window has been moved.

Private Sub *object*_**WindowMove( )**

Values

object
An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

See Also

WindowActivate, WindowResize

WindowResize Event

Occurs when the size of the Internet Explorer main window has changed.

Private Sub *object*_**WindowResize( )**

Values

object
An object expression that evaluates to an object in the Applies To list.

Applies To

WebBrowserApp

See Also

WindowActivate, WindowMove

Introduction

This document describes the mechanism of calling between Java and native (typically C or C++) code using the Microsoft VM for Java. This will include the native code DLL interface, as well as implications in the mechanism for garbage collection. This does not describe the Component Object Model (COM) interface which is another method of interfacing the Microsoft VM for Java with native code written in any other language which supports COM.

The purpose of this native code interface is to provide absolutely the fastest possible interface to native code. While using the COM model makes interfacing to the garbage collection system transparent to the programmer, this interface does not. Instead, it places the burden of interfacing with the garbage collection system on the programmer. If raw access to native code is not essential for performance reasons, using COM is the preferred method of accessing non-Java code.

Existing Native Code Interface

Typically when writing Java applications, developers may wish to use existing libraries of native code, or call out to routines written in native code for performance reasons. The following gives an example (using the current native code interface provided by in the Sun Java VM) as to how this would be done.

In Java, example syntax for declaring a method which is implemented in native code is as follows:

```
package myPackage;

class  myClass {
  public static native int myMethod  (myOtherClass  myObj);
}
```

This declares a static, native method named myMethod within the class myClass in the package myPackage. The method takes myObj (an object of type myOtherClass) as it's parameter.

This function can be then be called from within the Java code by invoking (for example):

```
...
int i;
myOtherClass myObj = new myOtherClass;

i = myClass.myMethod (myObj);
...
```

In order to deal with the interfacing between Java and C, there is a program called javah (included in the JDK) which takes the above **native** declaration and produces two pieces of code from it.

1. A C header file which can be used in C code which defines an appropriate prototype method and structure declaration for the containing class.
2. A stub function which is placed in the same file as the implementation.

The C native code implementation of the method, myMethod, would look something like:

```
int myPackage_myClass_myMethod (HObject *AlwaysNull, HmyOtherClass *myObj)
{
myOtherClass *mO = unhand (myObj);

  // body of the implementation of the function ...
// e.g. mO->member;
}
```

The function is defined with an initial first parameter (AlwaysNull) that is present only for historical reasons. It is always 0 in current Java VM implementations. Note that in the Microsoft Native Code Interface (below) we keep this convention for source code compatibility.

The stub (which javah generates) would look like this

```
Java_myPackage_myClass_myMethod_stub (ExecEnv *, paramblock * pPB);
```

When the function myMethod is called from Java, it is this stub which is called. The parameter block in

that stub is unwrapped, pushed onto the stack, and then the myPackage_myClass_myMethod function is called in C. The java code passes the parameter block from the frame, the stubs unwrap it, and then they pass it on to the actual function that you're interested in calling.

The final piece to calling is the **unhand** macro used in the C function above. This relies on the internal object representation (a double indirection between the myObj handle and the data - see below) to provide an actual pointer to the data in C. It is necessary to use the unhand macro in order to access the object data. It is recommended that unhand be used each time an object is accessed, as the object may have been garbage collected within Java, and a pointer reference to it obtained previously may then point to invalid data. Although existing VMs other than Microsoft's attempt to find and ensure validity of all unhanded object references in any C code which may have been called, there are cases where this attempt may not be sufficient to prevent pointers from becoming invalid.

How Object references are stored internally

The following diagram describes the internal storage details of objects within the existing native code system.

{ewc msdncd, EWGraphic, grpsweeper 1 /a "sweeper.bmp"}

Essentially, everywhere that a reference to an object is stored, there is a pointer to the handle which then references the object.

Implications for the existing native code interface

- An extra indirection (using the **unhand** macro) is required to get to every object referenced
- Despite the use of **unhand**, if you have an unhanded value referencing an object, that object may be moved by garbage collection, and the reference that you have could become invalid. To try and avoid this, when GC occurs under this model, the VM suspends all the threads that were running in Java code and tries to find out where the end of the native stacks are. It then conservatively(1) scans the entire C stack and register set of all threads for any DWORDS that might possibly point into objects. If any are found, these objects are not moved when the GC heap is compacted. Unfortunately, there are cases where this attempt could fail to find all outstanding unhanded references. It is also a somewhat time-consuming process.
- If there are many different pointers, it's extremely difficult for a compiler to enregister **unhanded** pointers (given restrictions on the numbers of registers in certain machine architectures). As a result, there will constantly be memory reads, which is generally less efficient.

Consequences

The main consequences from these implications are loss in performance with native code (which is one of the reasons why native code is used in the first place) and potentially non-deterministic errors when **unhand** has been used unsafely.

New Native Code interface

In the design of the native code interface in the Microsoft VM for Java, we have addressed many of the implications of prior interfaces as discussed above. Internally, object references are pointers directly to the object. The following diagram describes the internal storage details of objects within the Microsoft native code model.

{ewc msdncd, EWGraphic, grpsweeper 2 /a "sweeper.bmp"}

A pointer to our object is a pointer directly to the method table pointer in the object which is located just under the Data. Directly above the method table (which is a different data structure than current implementations support) is the data. When the Microsoft VM calls native code, it calls the target function directly without going through stubs located in the DLL library. There are stubs, but they are optimized, CPU specific code generated by the VM.

For source code compatability and header compatability, this native code model does support **unhand**. All the unhand does is cast the handle it to a pointer (and add 4 to its address). In the future, we will have another version of **unhand** (such as **ms_unhand**) which won't do any arithmetic (but should still be used), there will also be another header file generator which knows that there is this extra method table element at the base and will create appropriate C structures representing this. This will ensure that everything is fully accessible.

Since the Microsoft VM does not conservatively garbage collect native stacks, if any native code has outstanding pointer references during a garbage collection, the same problems as with the current native code model could be encountered. To remedy this while still providing maximum speed/minimum overhead, we have defined conventions to ensure a deterministic, low overhead solution with respect to outstanding native code pointers. These conventions rely on the fact that native code is responsible for enabling/disabling garbage collection as necessary, and informing the VM of locations in memory where object pointers exist.

Implications for exisiting code

This new native code interface is not binary compatible with the current interface. Existing Java applications that use the **native** keyword will need to consider the use they make of native code and how long they will be in native code functions. If you just call native code and return, it's OK to leave source code as it was in the previous model. If you're going to do block or take an undetermined amount of time to perform some task before returning to the VM, then you need to ensure that you allow GC. By default, garbage collection is turned off when you enter native code.

Implications and Benefits

- no extra level of indirection. Objects which are accessed don't require extra code to indirect through handles.

```
Benefit : performance improvement
```

- no explicit stubs

The target method is called directly from the VM. On the fly, the VM creates an assembler stub which sets up a frame appropriate for the calling convention of the current machine architecture. The process is as follows (the pushes are needed to transpose calling conventions)

setup native frame

push

push

push

call native method

In a way, this is similar to what happens in the existing native code interface, but it's carried out in a highly optimized form within the VM. This allows the VM to share stubs among similar methods and reduce working set, or optimize absolutely for speed. In either case, the native code libraries themselves can be smaller because they don't contain additional code for parameter unwrapping.

```
Benefit :  VM chooses optimizations, performance improvement.
```

- Using passed parameters is faster

```
Benefit : performance improvement
```

- This allows JIT code to be internally more optimized

```
Benefit : performance improvement
```

C code is just as portable as before. For the existing native code interface, there's less machine specific code within the VM to support the model, but there's always porting work required. Stubs are duplicated in every library (DLL or equivalent).

Functions for GC

In order to allow the control over the garbage collector by libraries using the new native code interface, the following functions are defined.

```
void enable_gc ();
```

If you're going to block for any length of time, or if you know you don't have any pointers that you care about, or those pointers have been wrapped in handles or GC frames, then you can call enable_gc(). This allows the GC to operate if necessary. If you're going to block, you're forced to wrap your pointers in handles or GC frames (see push_gc_frame and create_gc_handle below). Internally, enable_gc decrements the GC block count. If some thread calls enable_gc, it means that they're not preventing a garbage collection from occuring. However, a call to enable_gc doesn't necessarily cause garbage collection to occur.

```
void disable_gc ();
```

Increments GC block count and prevents GC from occurring.

```
BOOL check_gc ();
```

Returns TRUE if a GC occured during this call. This allows code to optimize its reloading of pointers. This can be used in strategic areas where code may be disabling GC for extended periods of time. Calling check_gc doesn't force a GC to occur, it just allows GC to happen if it is needed. Equivalent to decrement the GC block count, blocks if necessary to allow GC, then increments the block count.

```
DWORD get_gc_disable_count ();
```

Used for diagnostic purposes. Since enable_gc/disable_gc uses a counter, this provides a way to verify that the count is what is expected.

```
VOID push_gc_frame (CGCFrame *pGCF, VOID *pObjArr, DWORD size);
```

As mentioned above, pointers can be wrapped in handles or GCFrames. push_gc_frame packages up a number of pointers into a structure, which causes the pointers to be added to the root set for the garbage collector. This is a way of avoiding creating handles for them individually, so that whenever objects are moved due to garbage collection, the pointers in a CGCFrame will be updated to point to the new locations of the objects. CGCFrame is an abstract structure which is controlled by the VM.

```
VOID pop_gc_frame (CGCFrame);
```

Undoes the effect of the corresponding push_gc_frame. This is required to clear the stack of existing GC frames.

```
GCHANDLE create_gc_handle (Hobject *phObj);
```

Registers a single pointer with the garbage collector and returns a pointer to the registered pointer. The original variable passed is not automatically updated as objects are moved, but the updated value can be obtained by indirecting through the GCHANDLE.

```
VOID close_gc_handle (GCHANDLE gch);
```

Unregisters a handle created with create_gc_handle. This must be called in order to allow the garbage collector to free the object referenced by a handle.

GCFrames are nested on stack frames, and hence must be popped before you leave the stack frame. Handles are more long lived entities, but allow only for a single object reference per handle. The developer should select the most appropriate for the native code that is being written.

Goals

1. GC should be transparent as far as replacing malloc based C programs
2. GC overhead should be minimal (in terms of time and percentage execution) for most Java applets and applications.

Garbage Collection Mechanism

The key to countering the overhead for determining the difference between live and dead pointers is by making allocating memory very efficient. Most modern, high performance GCs start with a simple way of allocating memory. To allocate, we have a current block with a part that's used and a part that's free. The new object gets added on the end of the used pile and the pointer in the used pile is moved on. When we run off the end of the block, we allocate another block. The blocks are fixed size - if you need a larger block ("large" is dependent on tuning - typically this will be between 4k and 8k), then we allocate a custom block that will have only that object. All the blocks are linked together. The decision about the size of the block is the same tradeoff as page size within the operating system

To distinguish live pointers from dead pointers, we start from a root set where we know there are live pointers, and we follow those pointers by copying those into new blocks, then work through the each object that these pointers point to in these roots, and do the same with them, and so on. When we reach the end of scanning the new objects in the new copy, we know we've copied over every live object. This scheme can be fairly expensive if you always copy many objects that are very stable. We improve on this by taking a snapshot of the last point we GC'd and the new point we GC'd and copy only the objects that are in between - leaving old objects alone. This is called generational GC - using two generations in this case.

The large blocks are not copied - they're just logically moved from the old set to the new. We may (for version 2) we may make further improvements by considering only roots between the phases the part of memory that has been modified.

Garbage collection is called on allocation of new objects. When a new object is alloacted, the garbage collector detects memory threshold (a heuristic is used to decide if it needs to GC and, if so, sets a flag). The VM looks at the flag and initiates the process of checking the blockcount. We allow for one GC per namespace.

Mark and compact is not used as a garbage collection method, because they are inherently less efficient in situations where you have a lot of garbage - which will be typical in Java programs.

Conservative scanning or garbage collection is the term used when nothing is known about the data which is being checked. It may just be characters or some other simple data type, but if it looks like it may be a pointer, it is treated as such to err on the side of caution. Since the data is not known to be a pointer, it cannot be modified to refer to a new copy of the object to which it seems to point. Instead, the object in question must be fixed in memory and made immovable.

Introduction

While developed completely independently of one another, it is amazing how well Java integrates with COM. Often in computing, two diametrically different technologies will be forced to coexist by brute force. However, with COM and Java it turns out that while on the surface they are diametrically opposed, they, in fact are complementary to one another.

The following table is included to help the reader make a clear distinction between what Java and COM are. Because the technologies share common attributes, such as being object-oriented and general purpose, people have attempted to somehow compare the technologies as though they attempted to solve the same problem.

Attribute	Java	COM
Programming Language	Yes	No
Language Independent Component Architecture	No	Yes
Virtual Machine	Yes	No
Simple	Yes	Yes
Robust	Yes	Yes
Object-oriented	Yes	Yes
Platform Independent	Yes	Yes
General purpose	Yes	Yes
Multi-threaded	Yes	Yes
Distributed	No	Yes

Java is a great programming language for both implementing and using COM objects.

Java makes some of the more arcane and error prone aspects of COM programming easy. While Java is relatively new, it is already recognized as a great programming language. Enabling Java to create and use COM objects makes it a great programming language for COM.

COM makes Java a distributed language.

Every public Java class is a COM object and can be called remotely just like any other COM object. COM and DCOM are really the same thing—the real difference between COM and DCOM is the length of the wire between the two software components. Any COM object can be used remotely or locally; this is called location independence. Thus any COM object developed in Java can be executed on a remote server as well as on the client machine.

COM gives Java direct access to native code.

Any COM object appears to the Java programmer as a Java object. There is no need for large class libraries that wrap existing objects; the existing objects can simply be called directly.

Java and COM integrate together seamlessly.

The way that COM and Java are integrated by the Microsoft Java VM does not interfere or conflict with the "spirit" of Java in any way.

By integrating Java and COM existing technology investments are preserved while new technologies can be utilized. Java code can call, very naturally, into any ActiveX™ component, whether it's a control or Microsoft Excel. Likewise, any ActiveX™ component can call into any Java component. There is no need for new programming models and additional "layers" that technologies such as CORBA bring to the table.

Note Throughout this paper we refer to COM. When we do so we are including almost everything that has been known previously as OLE Automation. The fact is, OLE Automation was originally designed

to allow higher level programming languages access to COM objects. But over time more and more of the pieces of "OLE Automation" have been utilized by various OLE initiatives to the point where those pieces are not just being used for high-level language integration.

Java/COM Language Integration

The integration of Java and COM can be achieved by just making changes to the Java VM, and not adding any new keywords or constructs to the Java language. That is, the Java language already has the constructs that allow for the implementation and use of COM objects. In particular, Java, like COM, supports the notion of multiple interfaces on an object.

Key requirements for Java-COM integration include ease of programming and size/performance. Developing and using COM objects in Java is really no different than developing and using Java classes. The Microsoft Java implementation is also designed to be very efficient with minimal overhead per COM class created with Java.

The Java language has a certain philosophy related to it, and Microsoft does not wish to modify the language arbitrarily to integrate with COM in such a way that would be against the grain of that philosophy. Thus there are things that are possible with COM in C/C++ or other languages which are not possible with Java. Specific examples include restricting aggregation to a single aggregated object and a restrictive set of types that can be used in COM interface definitions. Appendix: Java/COM Language Integration Details contains a detailed discussion of exactly how COM and Java are integrated at the language level by Microsoft's Jakarta compiler.

Note Throughout this paper there are examples of Java source code illustrating how COM and Java are integrated. These examples use compiler features which will be found in Microsoft's Java compiler, codenamed Jakarta. It is important to note that the details of the language integration is completely up to the compiler, and other vendor's Java compilers may expose Java/COM integration differently. However, the way that the compilers produce Java .class files is standardized. This paper does not directly discuss the details of Microsoft's proposed .class file attributes for COM integration.

The following example illustrates how COM and Java are integrated showing how a COM object, which implements a simple "hello world" interface, is implemented in Java. Assume that the IHelloWorld interface is declared in a IDL (Interface Definition Language) file as:

```
[object, uuid(1F090040-9B7B-11cf-B63C-0080C792B782)]
interface IHelloWorld : IUnknown
{
  HRESULT SayHello();
};

[uuid(1F090041-9B7B-11cf-B63C-0080C792B782)]
coclass Hello
{
  interface IHelloWorld;
};
```

If the above IDL file were compiled into a COM type library named hello.tlb, the following Java code would provide an implementation of the hello COM class.

```
import samples.hello.*;
class Hello
  implements IHelloWorld
{
  void SayHello(void)
  {
    System.out.println("Hello world!");
```

```
  }
}
```

COM client code, written in any programming language could then create an instance of the Hello class and invoke the SayHello method. For example, the following C++ code does just this:

```
#include "hello.h"  // defines IHelloWorld and CLSID_Hello
IHelloWorld* pHello;
if (SUCCEEDED(CoCreateInstance(CLSID_Hello, NULL,
        CLSCTX_SERVER, IID_IHelloWorld, (void**)&pHello)))
{
  pHello->SayHello();
  pHello->Release();
}
```

As is evidenced by this example, developing COM objects in Java is no different than developing standard Java objects. In fact, all Java objects automatically *are* COM objects.

But what about using COM objects in Java? Just as all Java objects are COM objects, all COM objects appear to Java programmers as normal Java objects. To illustrate, here's some Java code which causes a Microsoft PowerPoint™ presentation to be run as a slide show:

```
powerpnt.Presentation pres = ActiveX.moniker.BindToObject("slides.ppt");
powerpnt.SlideShow show = pres.SlideShow();
show.Run(ppSlideShowFullScreen);
```

In this example a moniker was used to bind to a PowerPoint™ presentation. The ActiveX.moniker Java package is part of the classes included with Jakarta(1). Instead of using a class library function to activate a COM object, the Java new operator can be used as well:

```
powerpnt.Presentation pres = new powerpnt.Presentation();
...
```

Here, a new, empty presentation was created instead of loading an existing presentation.

There are several ways activate any COM object (even one implemented in Java!) remotely. One way is to use the COM library API's as exposed by the ActiveX Java class library:

```
samples.Hello.Hello hello = null;
hello = ActiveX.COM.CreateInstance(CLSID_HelloWorld, null,
CLSCTX_REMOTESERVER, "132.22.122.32");
hello.SayHello();
```

The example above will activate the COM object CLSID_Hello (which, in our example, was actually built with Java) on the remote machine identified with the IP address given. When the SayHello method is invoked, it will actually execute on this remote machine, and the message ("Hello World!") will be printed on a console on the remote machine.

As evidenced by the preceding series of examples, the integration of Java and COM is very tight. But most importantly, it is very natural to Java programmers.

Platform Integration

Existing Java applets, developed with the Sun JDK 1.0 are automatically exposed as ActiveX™ Controls by the Microsoft Java VM. ActiveX™ Controls is just a new name for OLE Controls. One implication of this is that any Java applet can now be plopped onto any application which supports ActiveX(tm) Controls, including Visual Basic, Access, Delphi, and Power Builder.

The fact that Java supports the creation of "thread-safe" objects, and because COM allows "thread-safe" objects to be executed remotely in very efficient manner makes Java an awesome tool for creating server applications.

Class Library Support

Microsoft will provide a set of classes which provide the equivalent functionality as provided by the COM library API, but are packaged and defined in such a way as to be very natural for Java programmers to use.

Java classes will be provided for each of the following major COM technologies:

- Core COM Library functionality. This maps roughly to the API's whose names are prefixed with "Co".
- Monikers.
- Structured storage.
- Utilities (e.g. StringFromGUID).

In addition, the Microsoft will provide classes which will make implementing various ActiveX™ components as easy as inheriting from a particular base class:

- ActiveX™ Controls and containers
- ActiveX™ document objects (DocObjs) and containers

Security Implications

Java applets running within a Web browser are restricted in several ways for security reasons. The term used to describe these restrictions is "sandboxing". These restrictions include not allowing applets to call into native code or to call Java classes which can not be verified by the Java bytecode verifier.

The ability to call into arbitrary COM objects, locally or remotely, from Java is clearly "calling native code" and thus is disabled for applets. However, through secure digital signatures, applets can be verified as coming from a trusted source, if an applet is digitally signed, it can make COM calls.

Note that these rules apply only to Java applets, not Java applications or other non-downloadable Java components.

Appendix: Java/COM Language Integration Details

This section details how COM and Java work together from the programming language perspective. The integration of Java and COM can be achieved by just making changes to the Java VM, and not adding any new keywords or constructs to the Java language. That is, the Java language already has the constructs that allow for the implementation and use of COM objects. In particular, Java, like COM, supports the notion of multiple interfaces on an object.

There are two perspectives from which Java-COM language integration can be viewed. The first is from the perspective of *using* or calling COM objects from Java code. The second is the *implementation* of COM objects in Java. Java-COM language integration is explored in depth in the sections below.

Calling COM Objects From Java

The following Java code snippet shows how you call Java classes in Java:

```
robocorp.bots.Robot robbie = new project.bots.Robot();
robbie.GoForward(10);
robbie.Turn(-90);
```

However, if Robot were actually implemented as a COM class (perhaps it was written using Visual Basic 4.0) the following code would be used:

```
robocorp.bots.Robot robbie = new robocorp.bots.Robot();
robbie.GoForward(10);
robbie.Turn(-90);
```

So what's the difference? There is none. And that's the point. Any COM object (with some minor restrictions described below) can be instantiated and called from Java as though it were a "real" Java implemented class.

Note Doing COM programming in Java is completely natural to the Java programmer.

Importing COM Class Information

In order to use COM objects and interfaces there needs to be some way to bring the names of those COM objects and interfaces into Java. In COM objects and interfaces are named by very large integers which are guaranteed to be unique—Globally Unique Identifiers—or GUIDs. The approach taken involves extending the compiler to directly understand the contents of a type library (which is exactly how COM has been integrated into Visual Basic). A type library is a essentially a database which describes the programming model of a component(2). The Java programmer will usually not refer to GUIDs (IIDs, CLSIDs, CATIDs) directly. Instead the he will refer to the names within a type library attached to those GUIDs. If the type referenced is a type library, the compiler will convert that type library to one or more .class files. The resulting .class files are tagged with special "attributes" which inform the Java VM that it is really talking to an external COM component, not a normal Java class.

To understand how this works, let's start with a short overview of how Java automatically handles compilation dependencies. Take the following import statement as an example:

```
import java.awt.Canvas;
```

The Java compiler will check along the CLASSPATH for a file called java/awt/Canvas.class. If found, it simply imports the symbols relating to java.awt.Canvas into the name space. The fully qualified name of Canvas is still java.awt.Canvas although it can be referred to by its alias Canvas. If the .class file is not found, the compiler searches for a java/awt/Canvas.java somewhere along the class path. If the .java file is found, the source is compiled into java/awt/Canvas.class and the .class file is imported.

Extending this to handle type libraries is done by extending the rules of this auto-compilation feature. Using the same example above, if java/awt/Canvas.java cannot be found, the compiler next searches for java/awt.tlb (.tlb being the standard filename extension of type library files) along the CLASSPATH. If the type library is found, the portion of the type library containing the information about the Canvas class is imported. If the type library is not found, the compiler attempts to build one based on an interface definition language (IDL) file it finds. Once the type information for a class has been successfully imported, the class and its interfaces are integrated into the name space and the programmer is able to access them just as if they were Java classes and interfaces.

Just as the import statement allows the programmer to import an entire package, an entire type library can be imported as well as only portions of the type library:

```
import robocorp.bots.*; // import all of bots type library
import robocorp.bots.Robot; // import just the Robot coclass
```

Development environments will also have an "Auto Import" dialog (much like Visual Basic's "References" dialog) that allows certain type libraries from the registry to be automatically imported for every compilation unit in the project. This dialog simply lists the type libraries in the registry and allows you to bring all components of each type library into your name space simply by checking a check mark next to the type library.

In order to support compilers which don't have the ability to deal with type libraries directly, Microsoft will provide a stand-alone tool which supports building the special .class files from a type library. This will invoke the subset of the compiler which performs the type library to .class file conversion.

Note The COM name space is tightly integrated with Java.

Reference Counting

In COM the lifetime of an object in memory is controlled by a mechanism known as reference counting. Each object maintains, internally, a count of the references clients have to it. If the reference count ever goes to zero, the component will delete itself from memory. Java, on the other hand, uses garbage collection to handle object lifetime. The Java VM, essentially, keeps track of the reference counts on objects and frees them when they are no longer needed.

The Microsoft Java VM garbage collector unifies these two approaches such that COM reference counting is handled automatically.

To illustrate, the example below, written in C++, shows how a client of an object uses the Release method to decrement the reference count of a COM object.

```
// Error handling ommitted for brevity
IRobot* probbie = NULL;
CoCreateInstance(CLSID_Robot, NULL, CLSCTX_SERVER, IID_IRobot,
(void**)&probbie);
probbie->GoForward(10);
probbie->Turn(-90);
probbie->Release();
```

In Java, the equivalent code would be:

```
import robocorp.bots.*;
Robot robbie = new Robot();
robbie.GoForward(10);
robbie.Turn(-90);
```

You'll notice that in the Java example we did not need to release the reference to the Robot object as we did in C++. This is because Java is a garbage collected language and can automatically detect when objects are no longer being used. The end result of this, with respect to COM, is that Java programmers do not have to worry about what is probably the most difficult aspect of COM programming: reference counting.

QueryInterface

But what about QueryInterface? COM objects can support multiple interfaces and the examples above only use a single interface (IRobot) on the object?

Java too supports the notion of multiple interfaces on objects. This fact alone is probably the number one reason that Java-COM integration works so well. To take the robot example above further, let's assume that the COM Robot class (CLSID_Robot) implements an interface called IRobotDiagnostics in addition to IRobot. Where IRobot is used to control what the robot does, IRobotDiagnostics allows for determining the status of the robot. Here's some C++ code where we make use of both interfaces. First we tell the robot to move forward 10 units and then to turn left 90 degrees using the IRobot interface. We then QueryInterface for IRobotDiagnostics and use that interface to tell the robot to display (or speak) a status message:

```
// Error handling ommitted for brevity
IRobot* probbie = NULL;
IRobotDiagnostics* probbieDiag = NULL;
CoCreateInstance(CLSID_Robot, NULL, CLSCTX_SERVER, IID_IRobot,
(void**)&probbie);
probbie->GoForward(10);
probbie->Turn(-90);
probbie->QueryInterface(IID_IRobotDiagnostics, (void**)&probbieDiag);
probbie->OutputStatusMessage();
probbieDiag->Release();
probbie->Release();
```

The same code, written in Java would look like this:

```
import robocorp.bots.*;
Robot robbie = new Robot();
IRobotDiagnostics robbieDiag = null;
robbie.GoForward(10);
robbie.Turn(-90);
robbieDiag = (IRobotDiagnostics)robbie;
robbieDiag.OutputStatusMessage();
```

It is the assignment statement that does the implicit QueryInterface for IRobotDiagnostics(3).

An explicit cast, like the example above, is necessary just as they are necessary in normal Java interfaces. An exception is thrown if the object does not support the desired interface just as happens with normal Java interfaces. The standard Java instanceof operator can also be used to determine up-front whether a given interface is supported.

Note Java makes several of the more arcane parts of COM programming simple.

Return Values, Error Handling, and Exceptions

All methods defined in a COM interface must return an HRESULT type. Note that the OutputStatusMessage in the IDL example above returns an HRESULT. Having methods return weird error codes like HRESULTs runs contrary to the philosophy of the Java language (as well as Visual Basic), therefore there must be a way to specify which parameter to a method is actually the return value. Fortunately COM already provides a mechanism for this: the retval keyword in IDL.

The way that OutputStatusMessage is declared above implies that it's return value is of type void. The HRESULT is important only in the case of an error. Therefore, in Java this method would be declared as:

```
void OutputStatusMessage(void);
```

If we wanted OutputStatusMessage to return a boolean indicating whether the status message was output or not, the correct IDL declaration would be:

```
HRESULT OutputStatusMessage([out, retval]VARIANT_BOOL* pRetVal);
```

This would be equivalent to the following Java declaration:

```
boolean OutputStatusMessage(void);
```

All this raises the issue of error handling and exceptions. Java has rich support for exception handling and exceptions are used consistently throughout Sun's class libraries. But COM doesn't directly support exception handling, so how can it interoperate seamlessly with Java? The key to answering this question is to understand the various ways that COM methods communicate (exceptional) failure information back to their callers.

The primary way is through the HRESULT that every COM interface method must return. The HRESULT is used to return either a success code or a failure code. In almost all cases there is only one success code defined (S_OK) but many failure codes. To illustrate here's the C++ example from above with error handling added:

```
try
{
  IRobot* probbie = NULL;
  HRESULT hr;
  hr = CoCreateInstance(CLSID_Robot, NULL, CLSCTX_SERVER, IID_IRobot,
(void**)&probbie);
  if (hr == S_OK)
  {
    hr = probbie->GoForward(10);
    if (hr == S_OK)
      hr = probbie->Turn(-90);
    probbie->Release();
  }

  if (hr != S_OK)
  {
    switch(hr)
```

```
    {
      case RPC_E_SERVERDIED:
        ThrowException("The communciations link to the robot failed");
      break;
    ...
      case E_UNEXPECTED:
      default:
        ThrowException("Unexpected error!");
      break;
    }
  }
}
catch(CException* e)
{
  ErrorMessage(e.szMsg);
}
```

This sample illustrates that even though it is not possible to throw exceptions across a COM call, it is possible to easily map the HRESULT return values to exceptions on the calling side. The Java VM does throw a native Java exception when a call to a COM interface method returns an HRESULT containing a failure code.

With this capability, Java code using COM objects can use Java exceptions naturally. The previous Java example is given below, but with error handling added:

```
import robocorp.bots.*;
...
try
{
  Robot robbie = new Robot();
  robbie.GoForward(10);
  robbie.Turn(-90);
}
catch(Throwable e)
{
  System.out.println("Something bad happened!");
}
```

The above example uses the "catch all" expression to catch all thrown exceptions. Usually programmers want to be able to catch specific exceptions and act upon them. Through a mechanism known as the IErrorInfo protocol, COM objects can expose much richer error information than can be conveyed in a single DWORD (the HRESULT). Visual Basic uses this mechanism to enable it's own peculiar style of exception handling for COM objects: the "On Error Goto" statement. For example, in Visual Basic you can do the following:

```
Dim robbie As New Robot
On Error Goto ErrorHandler
  robbie.GoForward(10)
  robbie.Turn(-90)
  goto TheEnd

ErrorHandler:
  'Err returns the code for the error
```

```
  Select Case Err
    Case RPC_E_SERVERDIED
      MsgBox " The communciations link to the robot failed "
            End

    'Catch all others
    Case Else
      MsgBox Error$
    End
  End Select

TheEnd:
  Set robbie = Nothing
```

If a COM method is called in Visual Basic and a failure code is detected, VB uses the IErrorInfo protocol to get rich error information from the object; in this example the Error$ string.

Microsoft's Java compiler and VM also use the IErrorInfo protocol to enable rich exception handling for calling COM objects. Here's the Java example:

```
import robocorp.bots.*;
...
try
{
  Robot robbie = new Robot();
  robbie.GoForward(10);
  robbie.Turn(-90);
}
catch(IRobotException e)
{
  System.out.println("Robbie had an error! " + e.GetWhyRobotHadAnError());
}
catch(IExcepInfo e)
{
  System.out.println("There was a general error! " + e.GetDescription());
}
```

Further below, were we discuss the implementation of COM objects in Java you will see how thrown Java exceptions get translated into the appropriate HRESULT/IErrorInfo facilities for non Java COM clients.

Note Java allows programmers to use exception handling when dealing with COM objects.

Properties

Java has no notion of object properties as defined by COM. For example, consider our IRobotDiagnostics interface:

```
[ object, uuid(6C6971D6-8E69-11cf-A54F-080036F12502), dual]
interface IRobotDiagnostics : IDispatch
{
...
  [propset] HRESULT TemperatureSampleFreq([in] long lFrequency);
  [propget] HRESULT TemperatureSampleFreq([out, retval] long* pTemp);
....
};
```

In Visual Basic the notion of properties on objects is directly supported, and thus the following code can be written to tell the robot to sample it's internal temperatures every 10 seconds:

```
Dim robbie As New Robot
Dim robbieDiag As IRobotDiagnostics
...
Set robbieDiag = robbie
robbieDiag.TemperatureSampleFreq = 10000
Debug.Print "Sample Frequency set to " + robbieDiag.TemperatureSampleFreq
Set robbieDiag = Nothing
Set robbie = Nothing
```

Without adding new language features to Java, the following is the only way to accomplish the same thing:

```
import robocorp.bots.*;
Robot robbie = new Robot();
IRobotDiagnostics robbieDiag = null;
...
RobbieDiag = (IRobotDiagnostics)robbie;
robbieDiag.set_TemperatureSampleFreq(10000);
System.out.println("Sample Frequency set to " +
Integer(robbieDiag.get_TemperatureSampleFreq()));
```

In other words the Java programmer can use COM properties but must do so though using method calls to set and get the values of those properties (C/C++ programmers have to do the same).

Creating Instances of COM Objects

In the examples above we've illustrated how Java's new operator can be used to create a new instance of a COM object. In effect when the class specified to the new operator is a COM class, new boils down to a CoCreateInstance call. However, COM supports other kinds of activation such as moniker binding. These additional activation, or binding, mechanisms are enabled through a standard set of classes that wrap the standard COM API. For example, to bind to a moniker in Java you would use code similar to the following:(4)

```
import robocorp.bots.*;
import microsoft.Moniker.*;
Moniker moniker = new Moniker("file:\\server\share\file.rob");
Robot robbie = moniker.BindToObject();
robbie.GoForward(50);
...
```

Implementing COM Objects In Java

The previous section discussed the *use* of COM objects in Java. This section discusses how COM objects can be implemented in Java.

Reference Counting and QueryInterface

As with using COM objects in Java, the Java language and VM hides all reference counting from the programmer who is implementing COM objects in Java. There is no need to ever implement AddRef() or Release() in Java.

Likewise, because the VM provides the implementation of IUnknown, there is no need to implement QueryInterface(); it is handled automatically.

Note Java greatly simplifies the implementation of COM objects.

Implementing COM Interfaces

Normally (in the absence of COM), when defining an implementing an interface on a Java class the interface being implemented is either declared within the code module (file) where the implementation occurs or in an external .class file which is imported via the import statement. It is illegal to both define an interface using the interface declaration *and* import the same interface's declaration from a .class file. The same rule holds true for COM interfaces. The implication is that you will never see a COM interface defined via Java. Instead, a COM interface is defined in IDL, compiled into a type library, and imported into Java as though it were a standard Java .class file.

To implement a given COM interface on a Java class simply use the implements modifier on the class declaration with a list of interface names and provide method bodies for each method in the interfaces (excluding QueryInterface, AddRef, and Release). For example, assuming the IRobot interface introduced in the previous section were described in IDL like this:

```
[ object, uuid(6C6971D5-8E69-11cf-A54F-080036F12502)]
interface IRobot : IUnknown
{
  ...
  HRESULT OutputStatusMessage([out, retval]BOOL* pRetVal);
  ...
};
```

And the IDL file were compiled into the robocorp/bots.tlb type library, the following Java code could be written to implement a robot object:

```
import robocorp.bots.*;
class RogerRobot implements IRobot
{
  ...
  boolean OutputStatusMessage(void)
  {
    return VoiceBox.Speak(CurrentStatus());
  }
  ...
}
```

Every Java class automatically implements IUnknown and IDispatch. The dispinterface implemented by the IDispatch that Java provides contains the public methods that are on the "default" interface for the class. For example the following Java class implements an IDispatch which supports the methods MethodA and MethodB:

```
class Example
{
  public void MethodA(void)
        { ... }

  public void MethodB(int foo)
  { ... }
}
```

At runtime the Microsoft Java VM will automatically provide type information for this IDispatch implementation (via IDispatch::GetTypeInfo so that clients can avoid the overhead of using IDispatch::GetIDsOfNames to do late binding.

Note Java objects are COM objects. No additional work is required.

Creating a COM Class Object

In all of the previous examples the classes defined support COM (e.g. they implement IUnknown and follow all the rules), but they are not necessarily creatable via COM (a "class object" in COM parlance) because no CLSID has been assigned to them. In order to make any piece of code be able to be activated by COM, that object must have a class identifier (CLSID) associated with it, and must support a class factory. In Java, this is accomplished by specifying a coclass that describes the COM related attributes of the class. The coclass attribute is part of the IDL language, and thus is stored in a type library. It allows a programmer to assign a CLSID to a particular class; for example:

```
// extract from MyStuff.IDL
...
[uuid(2CFB1F60-9150-11cf-B63C-0080C792B782]
coclass MyClass
{
  interface ISomeInterface;
  interface ISomeOther;
};
```

By using the import directive to import a type library containing such a coclass statement, and using the coclass name as the name of a Java class, the Java programmer is able to tell the Java compiler and VM that the Java class in question is a COM class object. The Java VM will automatically provide a class factory for the class. The following code illustrates this:

```
import project.MyStuff.*;
// The Java class, MyClass has a "coclass" statement in the MyStuff
// typelibrary imported above. The coclass specifies the class's
// CLSID as well as a base set of interfaces implemented by the
// class.
class MyClass implements IAnother
{
  // Method bodies for all of the methods in the
  // two interfaces specified in MyClass's coclass
  // (namely ISomeInterface and ISomeOther)
  ...

  // Method bodies for the IAnother interface
  ...
}
```

Note that the list of interfaces in the coclass statement does not restrict what interfaces an object actually implements. The example above (MyClass) illustrates this: it implements IUnknown, IDispatch, ISomeInterface, ISomeOther, and IAnother. The code for IUnknown and IDispatch are provided by the Java VM.

Therefore, by simply putting a coclass statement into a IDL file, compiling that IDL file into a type library using MIDL, importing the resulting type library into a Java compilation unit, and using the coclass name as the name of a Java class the programmer can create COM class objects. Client code, written in any programming language which supports COM can then use the objects.

Aggregation - Implementation Inheritance

Just as a Java class can singly inherit the implementation of another Java class using the extends modifier, any Java class can "inherit" the implementation of any (aggregatable) COM object. The syntax for specifying this is exactly as though the super-class were a Java class:

```
import com.microsoft.typelib.MyTypeLib
class MySubClass
  extends MyCoClass
{
  ...
  // Method1 is implemented by MyCoClass, we're overriding
  // the implementation here
  int Method1(void)
  {
    int n = super.Method1();
    if (n < 3)
      return n;
    return 4;
  }
  ...
}
```

As you can see from this example, MySubClass inherits all of the behavior of MyCoClass, which is a COM class object. Because of COM's binary standard, MyCoClass could be implemented in programming language, not just Java.

The super member variable of Java's root Object class works for COM super-classes just as it does for Java super-classes as illustrated by the implementation of the Method1override.

In the above example, the aggregate object exposes IUnknown and IDispatch; not any of the other interfaces that may be listed in the super class's coclass. Every COM interface that is to be exposed by the sub-class, even if implemented by the super-class, must be specified via the sub-class's implements modifier (or, if the suclass is a COM class, in its coclass in the type library). This allows a Java class implementor to effectively "filter out" interfaces that the sub-class (the aggregatee) implements which he does not want his super-class (the aggregate) to expose. Java allows the super-class implementation to override zero or more of the methods of any of the inherited interface implementations.

From a COM perspective, this support is implemented via a combination of aggregation and delegation. The implements modifier indicates to the VM the exact set of interfaces the outer object (the sub-class) should support via it's IUnknown::QueryInterface implementation. If the sub-class overrides one or more method of one of these interfaces, that interface will be exposed via delegation. If the sub-class does not override any of the methods of the interface then it will be exposed via aggregation.

All Java classes with at least one public method are exposed by the Java VM as aggregatable COM objects.

By Reference Parameters

COM interface methods can have parameters which pass "simple" types such as integers, floating point values, and characters, by reference. In most cases these parameters are "out" parameters. For example:

```
HRESULT Foo([in] long l, [out] long* Out1, [out, retval] long* retval);
```

In Java, it would seem like you could call this function like this: (5)

```
int A, B;
A = Foo(42, &B);
System.out.println("A = " + A + " and B = " + B);
```

However, Java does not support parameters which are references to the intrinsic data types, nor does it support pointers. So, how can a COM interface method like Foo be callable from Java?

The answer is the Java VM automatically translates functions like this to functions which take objects as parameters instead. For each of the intrinsic Java types (boolean, byte, char, short, int, float, long, and double) the Java VM provides a Java object equivalent. Thus the VM, treats the Foo method as though it were defined thus:

```
HRESULT Foo([in] long l, [in] ILong* Out1, [out, retval] long* retval);
```

Allowing the following Java code to be used:

```
int A;
Integer B;
A = Foo(42, B);
System.out.println("A = " + A + " and B = " + B.Value());
```

Notice that the variable B is of type Integer rather than int. Thus it's actually an object.

Limitations and Restrictions

There are some limitations and restrictions regarding COM-Java integration that should be made clear.

- As noted briefly above, there is a class of COM interfaces which cannot be called from nor implemented in Java. These interfaces are those that cannot be described in a type library. Note that this limitation is almost entirely in place because the current type library model is not rich enough to describe all interfaces. If and when this shortcoming is fixed the restriction on Java can be removed..
- Java classes can aggregate in only one COM class (because Java only supports single inheritance) while COM allows for multiple aggregates. It is possible that this limitation could be removed in the future, but it has been deemed not important enough to deal with at this time.

Microsoft is committed to delivering first class Java development tools, as well as the reference implementation of the Java virtual machine for Windows platforms. Microsoft's Component Object Model is the cornerstone of the ActiveX(tm) platform and by providing tight integration between Java and COM Microsoft is enabling customers and developers leverage hundreds of man years of effort put into development tools, training, applications, and other infrastructure while still being able to take advantage of new and exciting technologies such as Java. This paper discusses how Microsoft is integrating Java and COM.

Note This document is an early release of the final specification. It is meant to specify and accompany software that is still in devel-opment. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the func-tionality of the final specification or software. Microsoft assumes no responsibility for any damages that might occur either directly or indirectly from these inaccuracies. Microsoft may have trademarks, copyrights, patents or pending patent applications, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you a license to these trademarks, copyrights, patents, or other intellectual property rights.

(1)The names and final design of these classes is very preliminary and subject to change.

(2)Readers familiar with RPC systems such as DCE/RPC or CORBA can think of a type library as being a portable, binary representation of an interface definition language (IDL) file.

(3)This is very similar to how QueryInterface is supported in Visual Basic 4.0.

(4)Note that the design of the Microsoft COM Java classes (e.g. microsoft.Moniker in this example) is still under development.

(5)Again, remember that Java's int is 64 bits.

IUniformResourceLocator Interface Methods

This section provides information about IUniformResourceLocator interface methods.

IUniformResourceLocator::GetURL

IUniformResourceLocator::InvokeCommand

IUniformResourceLocator::SetURL

IUniformResourceLocator::GetURL

This method retrieves an object's URL.

HRESULT GetURL(
LPSTR **ppszURL* **);**

Parameters

ppszURL
Pointer to an LPSTR that will be filled-in with a pointer to the object's URL. Since this method allocates memory for the string, you must instantiate an IMalloc interface and free the memory using IMalloc::Free() when it is no longer needed. The following code fragment provides an example of how this can be done:

```
    //  START CODE FRAGMENT
    {
        // In this example, pURL is a global IUniformResourceLocator
pointer.
        LPSTR lpTemp;

        hres = pURL->GetURL(&lpTemp);
        if (SUCCEEDED(hres)){
            IMalloc* pMalloc;
            hres = SHGetMalloc(&pMalloc);
            if (SUCCEEDED(hres)){
                pMalloc->Free(lpTemp);
                pMalloc->Release();
            }
        }
    }
    //  END CODE FRAGMENT
```

Return Values

The function returns S_OK if the object's URL was retrieved successfully. If the object does not have a URL associated with it, the function returns S_FALSE and sets *ppszURL* to NULL. Otherwise, the return value is an error code that can be one of the following.

Value	Meaning
E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run.
URL_E_INVALID_SYTNAX	The URL's syntax in invalid.
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler.

IUniformResourceLocator::InvokeCommand

This method invokes a command on an object's URL.

HRESULT InvokeCommand(
PURLINVOKECOMMANDINFO *pURLCommandInfo;* **);**

Parameters

pURLCommandInfo
Pointer to a URLINVOKECOMMANDINFO structure that contains command information for the function.

Return Values

The function returns S_OK if the object's URL was opened successfully. If the object does not have a URL associated with it, the function returns S_FALSE. Otherwise, the return value is an error code that will be one of the following.

Value	Meaning
E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run.
URL_E_INVALID_SYTNAX	The URL's syntax in invalid.
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler.

IUniformResourceLocator::SetURL

This method sets an object's URL.

HRESULT SetURL(
LPCSTR *pcszURL*,
DWORD *dwInFlags* **);**

Parameters

pcszURL
Pointer to a const zero-terminated string that contains the URL to set. The protocol scheme may be included as part of the URL.

dwInFlags
Flag value that specifies the behavior for setting the protocol scheme. This field can contain one of the following values.

Value	Meaning
IURL_SETURL_FL_GUESS_PROTOCOL	If the protocol scheme is not specified in *pcszURL*, the system will automatically choose a scheme and add it to the URL.
IURL_SETURL_FL_USE_DEFAULT_PROTOCOL	If the protocol scheme is not specified in *pcszURL*, the system will add the default protocol scheme to the URL.

Return Values

The function returns S_OK if the object's URL was set successfully. Otherwise, the return value is an error code that will be one of the following.

Value	Meaning
E_OUTOFMEMORY	There is not enough memory to complete the operation.
IS_E_EXEC_FAILED	The URL's protocol handler failed to run.
URL_E_INVALID_SYTNAX	The URL's syntax in invalid.
URL_E_UNREGISTERED_PROTOCOL	The URL's protocol does not have a registered protocol handler.

Internet Shortcut Global Functions Overview

Remarks

This section contains information about the following Internet Shortcut global functions.

InetIsOffline

MIMEAssociationDialog

TranslateURL

URLAssociationDialog

InetIsOffline

This function determines whether or not the system is connected to the Internet

BOOL InetIsOffline (
DWORD *dwFlags* **);**

Parameters

dwFlags
Input flags for the function. This must be set to zero.

Return Values

The function returns TRUE if the local system in not currently connected to the Internet. The function returns FALSE if the local system is connected to the Internet or if no attempt has yet been made to connect to the Internet.

MIMEAssociationDialog

This function invokes the unregistered MIME content type dialog box.

HRESULT WINAPI MIMEAssociationDialog (
HWND *hwndParent*,
DWORD *dwInFlags*,
LPCSTR *pcszFile*,
LPCSTR *pcszMIMEContentType*,
LPSTR *pszAppBuf*,
UINT *ucAppBufLen* **);**

Parameters

hwndParent
The handle to the parent window of any posted child windows.

dwInFlags
Bit flag value that specifies the if an association is to be registered. The bit flag is the value MIMEASSOCDLG_FL_REGISTER_ASSOC (0x0001). If this bit is set, the selected application will be registered as the handler for the given MIME type. If this bit is clear, no association will be registered.

An application is only registered if this flag is set and the use indicates that a persistent association is to be made.

Registration is impossible if the string at *pcszFile* does not contain an extension.

pcszFile
Pointer to a null-terminated string that contains the name of the target file. This file must conform to the content type described by the *pcszMIMEContentType* parameter.

pcszMIMEContentType
Pointer to a null-terminated string that contains the unregistered content type.

pszAppBuf
Pointer to a buffer that will receive the path of the application specified by the user.

ucAppBufLen
The size of *pcszAppBuf*, in characters.

Return Values

The function returns S_OK if the content type was successfully associated with the extension. In this case, the extension is associated as the default for the content type and **pcszAppBuf** points to the string that contains the path of the specified application. The function returns S_FALSE if nothing was registered. Otherwise, the return value will be one of the following.

Value	Meaning
E_ABORT	The user canceled the operation.
E_FLAGS	The flag combination passed in *dwFlags* is invalid.
E_OUTOFMEMORY	There was insufficient memory available to complete the operation.
E_POINTER	One of the input pointers is invalid.

Note: This function does not validate the syntax of the input content type string at *pcszMIMEContentType*. A successful return value does not indicate that the specified MIME content type is valid.

TranslateURL

This function applies common translations to a given URL string, creating a new URL string.

HRESULT WINAPI TranslateURL(
LPCSTR *pcszURL*,
DWORD *dwInFlags*,
LPSTR **ppszTranslatedURL* **);**

Parameters

pcszURL
Pointer to the URL string to be translated.

dwInFlags
Bit flags that specify how the URL string is to be translated. This value can be a combination of the following.

TRANSLATEURL_FL_GUESS_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system will automatically choose a scheme and add it to the URL.

TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **TranslateURL**, the system will add the default protocol to the URL.

ppszTranslatedURL
Pointer variable that will receive the pointer to the newly created, translated URL string, if any. *ppszTranslatedURL is valid only if the function returns S_OK.

Return Values

The function returns S_OK upon success, and S_FALSE if the URL did not require translation. If an error occurs, the function returns a value that will be one of the following.

Value	Meaning
E_FLAGS	The flag combination passed in *dwFlags* is invalid.
E_OUTOFMEMORY	There was insufficient memory to complete the operation.
E_POINTER	One of the input pointers was invalid.

Remarks

This function does not validate the input URL string. A successful return value does not indicate that the URL strings are valid URLs.

URLAssociationDialog

This function invokes the unregistered URL protocol dialog box. This dialog allows the user to select an application to associate with a previously unknown protocol.

HRESULT WINAPI URLAssociationDialog(
HWND *hwndParent*,
DWORD *dwInFlags*,
LPCSTR *pcszFile*,
LPCSTR *pcszURL*,
LPSTR *pszAppBuf*,
UINT *ucAppBufLen* **);**

Parameters

hwndParent
Handle to the parent window.

dwInFlags
Bit flags that specify the behavior of the function. This value can be a combination of the following.

Value	Meaning
URLASSOCDLG_FL_USE_DEFAULT_NAME	Use the default file name (i.e. "Internet Shortcut").
URLASSOCDLG_FL_REGISTER_ASSOC	Register the selected application as the handler for the protocol specified in *pcszURL*. The application is only registered if this flag is set and the user indicates that a persistent association is desired.

pcszFile
Pointer to a const zero-terminated string that contains the file name to associate with the URL's protocol.

pcszURL
Pointer to a const zero-terminated string that contains the URL with an unknown protocol.

pszAppBuf
Pointer to a buffer that will receive the path of the application specified by the user.

ucAppBufLen
The size of *pcszAppBuf*, in characters.

Return Values

This function returns S_OK if the application is registered with the URL protocol. The function returns S_FALSE if nothing is registered. For example, the function returns S_FALSE when the user elects to perform a one-time execution via the selected application.

Internet Shortcut Enumerated Types Overview

Remarks

This section contains information on the following Internet Shortcut enumerated types.

IURL_SETURL_FLAGS

IURL_SETURL_INVOKECOMMAND_FLAGS

MIMEASSOCIATIONDIALOG_IN_FLAGS

TRANSLATEURL_IN_FLAGS

URLASSOCIATIONDIALOG_IN_FLAGS

IURL_SETURL_FLAGS

The following enumerated values are used with the IUniformResourceLocator::SetURL method. They are passed-in as the *dwInFlags* parameter.

```
typedef enum iurl_seturl_flags{
   IURL_SETURL_FL_GUESS_PROTOCOL         = 0x0001,
   IURL_SETURL_FL_USE_DEFAULT_PROTOCOL   = 0x0002
} IURL_SETURL_FLAGS;
```

Values

IURL_SETURL_FL_GUESS_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **IUniformResourceLocator::SetURL**, the system will automatically choose a scheme and add it to the URL.

IURL_SETURL_FL_USE_DEFAULT_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to **IUniformResourceLocator::SetURL**, the system will add the default protocol to the URL.

IURL_SETURL_INVOKECOMMAND_FLAGS

The following enumerated values are used in the **dwFlags** member of the URLINVOKECOMMANDINFO structure.

```
typedef enum iurl_invokecommand_flags{
   IURL_INVOKECOMMAND_FL_ALLOW_UI           = 0x0001,
   IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB   = 0x0002,
} IURL_INVOKECOMMAND_FLAGS;
```

Values

IURL_INVOKECOMMAND_FL_ALLOW_UI
If this bit is set, interaction with the user is allowed and the **hwndParent** member of the URLINVOKECOMMANDINFO structure is valid. If this bit is clear, interaction with the user is not allowed and the **hwndParent** member is ignored.

IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB
If this bit is set, the default verb for the Internet Shortcut's protocol is to be used and the **pcszVerb** member of the URLINVOKECOMMANDINFO structure is ignored. If this bit is clear, the verb is specified by **pcszVerb**.

MIMEASSOCIATIONDIALOG_IN_FLAGS

The following enumerated values are used with the MIMEAssociationDialog function to determine how it executes.

```
typedef enum mimeassociationdialog_in_flags{
    MIMEASSOCDLG_FL_REGISTER_ASSOC     = 0x0001
} MIMEASSOCIATIONDIALOG_IN_FLAGS;
```

Values

MIMEASSOCDLG_FL_REGISTER_ASSOC
Register the selected application will as the handler for the given MIME type. If this bit is clear, no association will be registered.

Remarks

An application is only registered if this flag is set and the user indicates that a persistent association is to be made.

TRANSLATEURL_IN_FLAGS

The following enumerated values are used with the TranslateURL function to determine how it will execute.

```
typedef enum translateurl_in_flags {
    TRANSLATEURL_FL_GUESS_PROTOCOL         = 0x0001,
    TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL   = 0x0002,
} TRANSLATEURL_IN_FLAGS;
```

Values

TRANSLATEURL_FL_GUESS_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to TranslateURL, the system will automatically choose a scheme and add it to the URL.

TRANSLATEURL_FL_USE_DEFAULT_PROTOCOL
If the protocol scheme is not specified in the *pcszURL* parameter to TranslateURL, the system will add the default protocol to the URL.

URLASSOCIATIONDIALOG_IN_FLAGS

The following enumerated values are used with the URLAssociationDialog function to determine how it will execute.

```
typedef enum urlassociationdialog_in_flags {
    URLASSOCDLG_FL_USE_DEFAULT_NAME  = 0x0001,
    URLASSOCDLG_FL_REGISTER_ASSOC    = 0x0002
} URLASSOCIATIONDIALOG_IN_FLAGS;
```

Values

URLASSOCDLG_FL_USE_DEFAULT_NAME
Use the default file name (i.e. "Internet Shortcut").

URLASSOCDLG_FL_REGISTER_ASSOC
Register the selected application as the handler for the protocol specified in the *pcszURL* parameter of URLAssociationDialog. The application is only registered if this flag is set and the user indicates that a persistent association is desired.

URLINVOKECOMMANDINFO

The URLINVOKECOMMANDINFO structure contains information for use with the IUniformResourceLocator::InvokeCommand method.

```
typedef struct  {
    DWORD    dwcbSize;
    DWORD    dwFlags;
    HWND     hwndParent;
    LPCSTR  pcszVerb;
} URLINVOKECOMMANDINFO, *PURLINVOKECOMMANDINFO;
```

Members

dwcbSize
Size of this structure, in bytes.

dwFlags
Specifies how the IUniformResourceLocator::InvokeCommand method will execute. This value can be a combination of the following.

Value	Meaning
IURL_INVOKECOMMAND_FL_ALLOW_UI	Interaction with the user is allowed and the **hwndParent** member of this structure is valid. If this is not set, interaction with the user is not allowed and the **hwndParent** member is ignored.
IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB	The default verb for the Internet Shortcut's protocol is to be used and the **pcszVerb** member is ignored. If this bit is not set, the verb is specified by **pcszVerb**.

hwndParent
Handle to the parent window. If **dwFlags** is set to IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB, this member is ignored.

pcszVerb
Pointer to a zero-terminated string that contains the verb to be invoked by IUniformResourceLocator::InvokeCommand. If **dwFlags** is set to IURL_INVOKECOMMAND_FL_USE_DEFAULT_VERB, this member is ignored.

Overview

This document introduces a new way to add scripting and OLE Automation capabilities to programs, such as applications or servers. With the advent of ActiveX™ Scripting, hosts can call upon disparate scripting engines from multiple sources and vendors to perform scripting between components. The script itself—language, syntax, persistent format, execution model, and so on—is left to the script vendor. Care has been taken to allow arbitrary language "back-ends" to be used by hosts relying on ActiveX Scripting to perform these tasks. The terms *script* and *scripting* are used throughout this document to mean an executable blob, whether that be a piece of text, a block of pcode, or even machine-specific execution byte codes.

ActiveX Scripting Background

ActiveX Scripting components can be grouped into two major categories: ActiveX Scripting hosts and ActiveX Scripting engines. A host creates a scripting engine to allow scripts to be run against the host. Examples of (potential) ActiveX Scripting hosts include:

- Microsoft® Internet Explorer
- Internet authoring tools
- Tarantula/Gibraltar (server-based scripting)
- Shell

ActiveX Scripting engines can be developed for any desired language/runtime environment, including:

- Microsoft® Visual Basic™ for Applications (VBA)
- Microsoft® Visual Basic™ Scripting Edition (VBScript)
- Perl
- Lisp, Scheme

To make implementation of the host as flexible as possible, an OLE Automation wrapper for ActiveX Scripting will be provided. However, hosts that instantiate the scripting engine via this OLE Automation wrapper object will not have the control over the runtime namespace, the persistence model, and so on, that they would have if ActiveX Scripting were written to directly.

Interface elements required only in an authoring environment are isolated in this design, so that nonauthoring hosts (such as browsers and viewers) and script engines (for example, VBScript) can be kept lightweight.

Basic Architecture

The illustration below shows the interaction between an ActiveX Scripting host and an ActiveX scripting engine.

{ewc msdncd, EWGraphic, grpsweeper 3 /a "sweeper.bmp"}

The following list annotates the steps involved in the handshake (the actual nesting of the calls has been omitted for clarity):

1. **Create Project**. The host will load a project/document as it sees fit. (This is not really a step particular to ActiveX Scripting but has been included here for completeness.)
2. **Create the ActiveX Scripting Engine**. The host calls **CoCreateInstance()** to create a new ActiveX Scripting engine, specifying a **CLSID** to identify the specific scripting engine to be used. The HTML browser, by way of example, will be given this class id through the **PROGID=** attribute on the HTML OBJECT tag.
3. **Load the Script**. If the script contents have been previously persisted, the host then calls the script engine's **IPersist*::Load()** method to feed it the script storage, stream, or property bag. Otherwise, **IPersist*::InitNew()** or **IActiveScriptParse::InitNew()** is used to create a null script. Hosts maintaining a script as text can use **IActiveScriptParse::ParseScriptText()** to feed the scripting engine the text of the script, after calling **InitNew()**.
4. **Add Items**. For each top-level named entity imported into the scripting engine's namespace (such as pages and forms), the host calls **IActiveScript::AddNamedItem()** to create an entry in the engine's namespace. This step is not necessary if such items are already part of the persistent state of the script loaded in step 3. Sublevel named items (such as Controls on an HTML page) are not added via **AddNamedItem()**, but rather, are obtained indirectly from top-level items through **ITypeInfo** and **IDispatch**.
5. **Run the Script**. The host calls the **IActiveScript::SetScriptState(STRIPTSTATE_CONNECTED)**. This call would likely perform any scripting engine construction work, including static bindings, hooking up to events (see below), and executing something similar to a scripted "**main()**" function.
6. **Get Item Information**. Each time the script engine needs to associate a symbol with a top-level item, it will call the **IActiveScriptSite::GetItemInfo()** method. This method returns information about the object corresponding to the name passed in.
7. **Hookup Events**. In the final phase before starting the actual script, the scripting engine will connect to the events of all of the relevant objects via **IConnectionPoint** and other methods.
8. **Invoke Properties/Methods**. As the script runs, references to methods and properties on named objects will be realized by the scripting engine through **IDispatch::Invoke()** or other standard OLE binding mechanisms.

The following terms are used to refer to entities in the above diagram:

Host	The application or program that owns the ActiveX Scripting engine. The host implements **IActiveScriptSite** and optionally **IActiveScriptSiteWindow**.
Scripting Engine	The OLE object that implements **IActiveScript** and optionally **IActiveScriptParse**.
Script	The data that makes up the "program" that the scripting engine runs. The script gets loaded into the scripting engine via **IPersist*** or **IActiveScriptParse**.
Scriptlet	A portion of a script that gets attached to an object via **IActiveScriptParse**. The aggregate collection of scriptlets is the script.
Script Language	The language (for example, VBScript) and the semantics thereof.
Named Item	COM object (OLE Automation support is preferable) that the host deems interesting to the script. Examples include the HTML Page and Browser in a Web

	Browser, and the Document and Dialogs in Microsoft Word.
Code Object	An instance created by the scripting engine that is associated with a named item, such as the module behind a Form in Visual Basic, or a C++ class associated with a named item. Preferably, this code object is an OLE COM object supporting OLE Automation so that the code object can be manipulated by the host or another nonscript entity.

The ActiveX Scripting Engine

To write an ActiveX Scripting engine, write an OLE COM object supporting the interfaces listed below:

Interface	Required?	Description
IActiveScript	Yes	Basic scripting ability.
IPersist*	Yes. At least one of the following(*)	Persistence support.
	IPersistStorage	**DATA=***{url}* syntax for **OBJECT** tag.
	IPersistStreamInit	Same as above, as well as **DATA=***"string-encoded byte stream"* syntax for **OBJECT** tag.
	IPersistPropertyBag	**PARAM=** syntax for **OBJECT** tag.
IActiveScriptParse	No	Ability to add script text, evaluate expressions, and so on.

Support for the **IActiveScriptParse** interface is optional; however, if it is not supported, the script engine must implement some sort of **IPersist*** interface in order to load a given script.

Registry Standard

An ActiveX Scripting engine can identify itself using Component Categories. ActiveX Scripting currently defines two Component Categories:

CATID_ActiveScript	Indicates the **CLSID**s are ActiveX Scripting engines that support, at a minimum, **IActiveScript** and a persistence mechanism (**IPersistStorage**, **IPersistStreamInit**, or **IPersistPropertyBag**).
CATID_ActiveScriptParse	Indicates the **CLSID**s are ActiveX Scripting engines that support, at a minimum, **IActiveScript** and **IActiveScriptParse**.

Although **IActiveScriptParse** is not a true persistence mechanism, it does support an **InitNew**() method which is functionally equivalent to **IPersist*::InitNew**().

Script Engine States

An ActiveX Scripting engine can be in one of several general states:

Uninitialized	The script has not yet been initialized or loaded using an **IPersist** interface, or does not have an **IActiveScriptSite**set. The scripting engine is generally not usable from this state until steps are taken to initialize the scripting engine.
Initialized	The script has been initialized with an **IPersist** interface and has an **IActiveScriptSite** set, but is not connected to host objects and sinking events. Note that this state simply means that **IPersist*::Load()**, **IPersist*::InitNew()**, or **IActiveScriptParse::InitNew()** has been completed, and **IActiveScript::SetScriptSite()** has been called. No code can run in this mode. Code that is executed via **IActiveScriptParse::ParseScriptText()** is queued and executed after the transition to the **started** state.
Started	The transition from the **initialized** state to **started** state causes any code that was queued in the **initialized** state to execute. When in this state, code can execute, but the scripting engine is not connected to events of the objects added via **IActiveScript::AddNamedItem()**. Thus, code can be executed by calling **IDispatch** obtained from **IActiveScript::GetScriptDispatch()** or by calling **IActiveScriptParse::ParseScriptText()**. It is possible that further background initialization (progressive loading) is still ongoing, and that calling **SetScriptState(SCRIPTSTATE_CONNECTED)** may cause the script to block until initialization is complete.
Connected	The script is loaded and connected for sinking events from host objects.
Disconnected	The script is loaded and has a runtime state, but is temporarily disconnected from sinking events from host objects. This state is distinguished from the **initialized** state in that the transition to this state does not cause the script to reset, the runtime state of the script is not reset, and a script initialization procedure is not executed
Closed	The script has been closed. The scripting engine will no longer work and will return errors for most methods.

These states are entered via the following methods:

{ewc msdncd, EWGraphic, grpsweeper 4 /a "sweeper.bmp"}

The scripting engine performs the following actions on the given transitions:

{ewc msdncd, EWGraphic, grpsweeper 5 /a "sweeper.bmp"}

A Word About Threading

Because an ActiveX Scripting engine can be used in many environments, it is important to keep its execution model as flexible as possible. For example, server-based hosts may have a multithreaded design that they want to preserve while using ActiveX Scripting in a efficient manner. At the same time, hosts that do not care about threading, such as a typical application, should not be burdened with threading management. ActiveX Scripting achieves this balance by restricting the ways a free-threaded scripting engine may call back to the host, freeing hosts from this burden.

For Scripting Engine Implementers

Scripting engines used on servers are typically implemented as free-threading COM objects. This means that methods on **IActiveScript** (and its associated interfaces) may be called from any thread in the process, without marshaling(**). Synchronization is the responsibility of the scripting engine. For scripting engines that are not internally reentrant, or for language models that are not multithreaded, synchronization could be as simple as serializing access to the scripting engine with a mutex. Of course, certain member functions, such a **InterruptScriptThread()**, should not be serialized in this way, so that a stuck script can be terminated from another thread.

The fact that **IActiveScript** is free-threaded generally implies that **IActiveScriptSite** and the host's object model should be free-threaded as well. This would make implementation of the host quite difficult, particularly in the common case where the host is a single-threaded Windows application with single-threaded or apartment-model OLE Controls in its object model. For this reason, the following constraints are placed on the scripting engine's use of **IActiveScriptSite:**

Scripting Engine Rules:

- The script site will always be called in the context of a host thread. That is, the scripting engine will never call the script site in the context of a thread that it created, but only from within a scripting engine method that was called from the host (directly via **IActiveScript** and its derivatives, through the exposed scripting engine's dispatch object, or via a Windows message; indirectly from an event source in the host's object model).
- The script site will never be called from within the context of a simple thread state control method (for example, **InterruptScriptThread()**) or from the **Clone()** method.

For Scripting Hosts

The host can safely assume that **IActiveScriptSite** will only be called in the context of the base thread, as long as it obeys the following rules:

Scripting Host Rules:

- Choose a base thread (generally the thread with the message loop);
- Instantiate the scripting engine in this base thread;
- Call scripting engine methods only from this base thread, except where specifically allowed, as in the cases of **InterruptScriptThread()** and **Clone()**;
- Call the scripting engine's dispatch object only in this base thread;
- Ensure that the message loop runs in this base thread if an **HWND** is provided; and
- Ensure that objects in the host's object model only source events in this base thread.

Note that these rules are automatically followed by all single-threaded hosts. The restricted model described above is intentionally loose enough to allow a host to abort a stuck script by calling **InterruptScriptThread()** from another thread (initiated by a CTRL-BREAK handler or the like), or to duplicate a script in a new thread using **Clone()**.

Note that none of these restrictions apply to a host that chooses to implement a free-threaded **IActiveScriptSite** and a free-threaded object model. Such a host may use **IActiveScript** from any thread, without restriction.

Script Thread Execution States

Each script thread(***) can be in one of several execution states:

NotInScript	The thread is not currently running in a script (sinking a scripted event, processing **ParseScriptText()**, or being called through a global script function published through **GetScriptDispatch()**).
Running	The thread is currently executing script code.

IActiveScript

The scripting engine must implement this interface in order to be an ActiveX Scripting engine.

```
Interface IActiveScript : IUnknown
  {
  HRESULT SetScriptSite([in] IActiveScriptSite *pScriptSite);
  HRESULT GetScriptSite([in] REFIID iid, [out] void **ppvScriptSite);
  HRESULT SetScriptState([in] SCRIPTSTATE   ss);
  HRESULT GetScriptState([out] SCRIPTSTATE *pss);
  HRESULT Close();
  HRESULT AddNamedItem([in] LPCOLESTR pstrName, [in] DWORD dwFlags);
  HRESULT AddTypeLib([in] REFGUID guidTypeLib, [in] WORD wMaj, [in] WORD
wMin, [in] DWORD dwFlags);
  HRESULT GetScriptDispatch([in] LPCOLESTR pstrItemName, [out] IDispatch
**ppdisp);
  HRESULT GetCurrentScriptThreadID([out] SCRIPTTHREADID *pstid);
  HRESULT GetScriptThreadID([in] DWORD dwWin32ThreadID, [out]
SCRIPTTHREADID *pstid);
  HRESULT GetScriptThreadState([in] SCRIPTTHREADID stid, [out]
SCRIPTTHREADSTATE *psts);
  HRESULT InterruptScriptThread([in] SCRIPTTHREADID stid, [in] EXCEPINFO
*pei, [in] DWORD dFlags);
  HRESULT Clone([out] IActiveScript **ppscript);
  };
```

SetScriptSite

Informs the scripting engine of the **IActiveScriptSite** site provided by the host. This method must be called before any other **IActiveScript** methods may be used.

HRESULT IActiveScript::SetScriptSite(
[in] IActiveScriptSite **pScriptSite*
);

Parameters

pScriptSite
The host-supplied script site to be associated with this instance of the scripting engine. This site must be uniquely assigned to this scripting engine instance; it cannot be shared with other scripting engines.

Returns

S_OK	Success.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.
E_FAIL	Unspecified error. The scripting engine was unable to complete its site initialization.
E_UNEXPECTED	The call was not expected. (for example, a site has already been set.)

GetScriptSite

Retrieves the site object associated with the ActiveX Scripting engine.

HRESULT IActiveScript::GetScriptSite(
[in] REFIID *iid*,
[out] void ***ppvSiteObject*
);

Parameters

iid
The IID of the requested interface.

ppvSiteObject
The returned interface pointer to the site object.

Returns

S_OK	Success.
S_FALSE	Success, but no site has yet been set, so **ppvSiteObject** is NULL.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.
E_NOINTERFACE	The specified interface is not supported.

GetScriptState

Returns the current state of the scripting engine. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

HRESULT IActiveScript::GetScriptState(
[out] SCRIPTSTATE **pss*
);

Parameters

pss
Returned state of the indicated thread, as follows:

SCRIPTSTATE_UNINITIALIZED	The script has just been created and has not yet been initialized using an **IPersist** interface and **IActiveScript::SetScriptSite()**.
SCRIPTSTATE_INITIALIZED	The script has been initialized, but is not running (connecting to other objects, sinking events) or executing any code. Code can be queried for execution via **IActiveScriptParse::ParseScriptText()**.
SCRIPTSTATE_STARTED	The script can execute code, but is not yet sinking the events of objects added via **IActiveScript::AddNamedItem()**.
SCRIPTSTATE_CONNECTED	The script is loaded and connected for sinking events.
SCRIPTSTATE_DISCONNECTED	The script is loaded and has a runtime execution state, but is temporarily disconnected from sinking events.
SCRIPTSTATE_CLOSED	The script has been closed. The scripting engine will no longer work and will return errors for most methods.

Returns

S_OK	Success.
E_POINTER	An invalid pointer was passed.

SetScriptState

Puts the scripting engine into the given state. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

HRESULT IActiveScript::SetScriptState(
[in] SCRIPTSTATE *ss*
);

Parameters

ss

Sets the scripting engine to the given state:

SCRIPTSTATE_INITIALIZED — Returns the scripting engine back to the **initialized** state (from **started**, **connected**, or **disconnected**). Because languages can vary widely in semantics, scripting engines are not required to support this state transition. (Engines that support **IActiveScript::Clone()** must, however, support this state transition.) Hosts must prepare for this and take the equivalent action: **Release()** the current scripting engine, create a new scripting engine, and **Load()** or **InitNew()** (and possibly also call **ParseScriptText()**). Use of this transition should be considered an optimization of the above steps. Note that any information the scripting engine has obtained about the names of Named Items and the type information describing Named Items remains valid.

Because languages vary widely, defining the exact semantics of this transition is difficult. Minimally, the scripting engine must disconnect from all events, and release all of the **SCRIPTINFO_IUNKNOWN** pointers obtained via **IActiveScriptSite::GetItemInfo()**. These pointers must be refetched after the script is run again. The scripting engine should also reset the script back to an initial state that is appropriate for the language. VBScript, for example, will reset all variables, and retain any code added dynamically via **IActiveScriptParse** with the **SCRIPTTEXT_ISPERSISTENT** flag set. Other languages may choose to retain current values (such as Lisp because there is no code/data separation) or reset to a well-known state (this includes languages with statically initialized variables). These languages may or may not retain code added via **IActiveScriptParse**.

Note that the transition to the **started** state should have the same semantics (that is, it should leave the scripting engine in the same state) as **IPersist::Save()**-ing the scripting engine and then **IPersist::Load()**-ing a new scripting engine; these actions should have the same semantics as **IActiveScript::Clone()**. Scripting engines that do not yet support **Clone()** or **IPersist*** should carefully consider how the transition to the **started** state should behave, so that such a transition would not violate the above conditions if **Clone()** or **IPersist*** support were later added.

During this transition to the **started** state, the scripting engine will disconnect from event sinks after the appropriate destructors, and

	so on, are executed in the script. To avoid having these destructors executed, the host can first move the script into the **disconnected** state before moving into the **started** state. Use **InterruptScriptThread()** to cancel a running script thread without waiting for current events, and so on, to finish running.
SCRIPTSTATE_STARTED	The transition to this mode causes any code that was queued during the **initialized** state to be executed. From this state, script code may be executed, for example, by calling **IActiveScriptParse::ParseScriptText()** or by calling the **IDispatch** obtained from **IActiveScript::GetScriptDispatch()**. The transition to this state is also the appropriate time to execute routines such as a **main()**-like script routine, if appropriate for the script language.
SCRIPTSTATE_CONNECTED	Causes the script to connect to events. If this is a transition from the **initialized** state, the scripting engine should transition through the **started** state, performing the necessary actions, before entering the **connected** state and connecting to events.
SCRIPTSTATE_DISCONNECTED	Causes the script to disconnect from event sinks. This can be done either *logically* (ignoring events received), or *physically* (calling **Unadvise()** on the appropriate connection points). Returning to the **connected** state reverses this process. If this is a transition from the **initialized** state, the scripting engine should transition through the **started** state, performing the necessary actions, before entering the **disconnected** state. Event sinks that are in progress are completed before the state changes (use **InterruptScriptThread()** to cancel a running script thread). The script's execution state is maintained. For example, an HTML browser may put the scripting engine into this state when a scripted HTML page is moved into the LRU cache, before the page is actually destroyed.

Returns

S_OK	Success, the script has entered the given state.
S_FALSE	Success, but the script was already in the given state.
OLESCRIPT_S_PENDING	The function was queued successfully, but the state hasn't changed yet. When the state changes, the site will be called back on **IActiveScriptSite::OnStateChange()**.
E_FAIL	The scripting engine does not support the transition back to the **initialized** state. The host must discard this scripting engine and create, initialize, and load a new scripting engine to achieve the same effect.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).

Close

Causes the scripting engine to abandon any currently loaded script, lose its state, and release any interface pointers it has to other objects, thus entering a **closed** state. Event sinks, immediately executed script text, and macro invocations that are already in progress are completed before the state changes (use **InterruptScriptThread()** to cancel a running script thread). This method must be called by the creating host before it calls **Release()** to prevent circular reference problems.

HRESULT IActiveScript::Close( );

Returns

S_OK	Success, the script has been closed.
S_FALSE	Success, but the script was already closed.
OLESCRIPT_S_PENDING	The function was queued successfully, but the state hasn't changed yet. When the state changes, the site will be called back on **IActiveScriptSite::OnStateChange()**.
E_UNEXPECTED	The call was not expected (for example, the scripting engine was already in the **closed** state).

AddNamedItem

Adds a root-level name (an object with properties/methods, an event source, or both) to the scripting engine's name space.

HRESULT IActiveScript::AddNamedItem(
[in] LPCOLESTR *pstrName*,
[in] DWORD *dwFlags*
);

Parameters

pstrName
Name of the item as viewed from the script. Must be unique and persistable.

dwFlags
Flags associated with item. The following flags are defined:

SCRIPTITEM_ISPERSISTENT	Indicates that the item should be saved if the scripting engine is saved. Similarly, setting this flag indicates that a transition back to the **initialized** state should retain the item's name and type information (the scripting engine must, however, release all pointers to interfaces on the actual object).
SCRIPTITEM_ISSOURCE	Indicates that the item sources events that the script can sink. Children (properties of the object that are in themselves objects) can also source events to the script. This is not recursive, but provides a convenient mechanism for the common case, for example, of adding a container and all of its member controls.
SCRIPTITEM_ISVISIBLE	Indicates that the item's name is available in the namespace of the script, allowing access to the properties, methods, and events of the item. Because by convention, the properties of the item include the item's children, all child object properties and methods (and their children, recursively) will be accessible.
SCRIPTITEM_GLOBALMEMBERS	Indicates that the item is a collection of global properties and methods associated with the script. Normally, a scripting engine would ignore the object name (other than for the purpose of using it as a cookie for **IActiveScriptSite::GetItemInfo()**, or for resolving explicit scoping) and expose its members as global variables and methods. This allows the host to extend the library (runtime functions and so on) available to the script. It is left to the scripting engine to deal with name conflicts (for example, when two **SCRIPTITEM_GLOBALMEMBERS** items have methods of the same name), although an error should not be returned because of this situation.
SCRIPTITEM_NOCODE	Indicates that the item is simply a name being added to the script's name space, and should not be treated as a item for which code should be associated. For example, without this flag being set, VBScript will create a separate module for the named item, and C++ might create a separate wrapper class for the named item.
SCRIPTITEM_CODEONLY	Indicates that the named item represents a code-only object, and that the host has no IUnknown to be associated with this code-only object. The host only has a name for this object. In object-oriented

(C++-like) languages, this would create a class. Not all languages support this flag.

Returns

S_OK	Success, named item has been added to the script's namespace.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.

AddTypeLib

Adds a type library to the namespace for the script. This is similar to **#include** in C/C++. It allows a set of predefined items (class definitions, typedefs, named constants, and so on) to be added to the runtime environment available to the script.

HRESULT IActiveScript::AddTypeLib(
[in] REFGUID *guidTypeLib***,**
[in] DWORD *dwMaj***,**
[in] DWORD *dwMin***,**
[in] DWORD *dwFlags*
);

Parameters

guidTypeLib
LIBID of the Type library to add.

dwMaj
Major version number.

dwMin
Minor version number.

dwFlags
Option flags:

SCRIPTTYPELIB_ISCONTROL	Indicates that the type library describes an OLE Control used by the host.

Returns

S_OK	Success, the specified type library has been added.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).
TYPE_E_CANTLOADLIBRARY	The specified type library could not be loaded.
E_INVALIDARG	An invalid argument was passed.

GetScriptDispatch

Returns the **IDispatch** for methods/properties associated with the running script itself. If NULL is passed for pstrItemName, this object contains as its members all of the global methods and properties defined by the script. Through this interface (and its associated **ITypeInfo**), the host can invoke script methods or view/modify script variables. Note that because methods and properties can be added via **IActiveScriptParse**, and so on, the **IDispatch** returned may dynamically support new methods and properties. Similarly, **IDispatch::GetTypeInfo()** should return a new, unique **ITypeInfo** when methods and properties are added. Note, however, that language engines must not change the **IDispatch** in an incompatible way with previous TypeInfos returned. That implies, for example, that **DISPID**s will never be reused.

HRESULT IActiveScript::GetScriptDispatch(
[in] LPCOLESTR pstrItemName [out] IDispatch ***ppdisp*
);

Parameters

pstrItemName
The Named Item for which the caller wants the associated dispatch object. NULL indicates a global dispatch object, if supported.

ppdisp
The returned object associated with the script's global methods/properties. If the scripting engine does not support such an object, **NULL** is returned.

Returns

S_OK	Success.
S_FALSE	The scripting engine does not support a dispatch object. **NULL** is returned in **ppdisp**.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.

GetCurrentScriptThreadID

Returns a scripting-engine-defined identifier for the currently executing thread. This identifier can then be used in subsequent calls to script thread execution control methods (for example, **InterruptScriptThread()**, and so on).

HRESULT IActiveScript::GetCurrentScriptThreadID(
[out] SCRIPTTHREADID **pstidThread*
);

Parameters

pstidThread
Returned script thread ID associated with the current thread. The interpretation of this identifier is left to the scripting engine, and may in fact, just be a copy of the Windows thread id. Note that if the Win32 thread terminates, this ID becomes unassigned and may subsequently be assigned to another thread. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

Returns

S_OK	Success.
E_POINTER	An invalid pointer was passed.

GetScriptThreadID

Returns a scripting-engine-defined identifier for the currently executing thread. This identifier can then be used in subsequent calls to script thread execution control methods (for example, **InterruptScriptThread()**, and so on).

HRESULT IActiveScript::GetScriptThreadID(
[in] DWORD *dwWin32ThreadID*,
[out] SCRIPTTHREADID **pstidThread*
);

Parameters

dwWin32ThreadID
The thread ID of a running Win32 thread in the current process. Use **GetCurrentThreadID()** for this parameter to get the thread ID of the currently executing thread.

pstidThread
Returned script thread ID associated with the indicated Win32 thread. The interpretation of this identifier is left to the scripting engine, and may in fact, just be a copy of the Windows thread id. Note that if the Win32 thread terminates, this ID becomes unassigned and may subsequently be assigned to another thread. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

Returns

S_OK	Success.
E_POINTER	An invalid pointer was passed.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).

GetScriptThreadState

Returns the current state of a script thread. Note that if this is not the current thread, the state may change at any time. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

HRESULT IActiveScript::GetScriptThreadState(
[in] SCRIPTTHREADID *stidThread*,
[out] SCRIPTTHREADSTATE **pstsState*
);

Parameters

stidThread

Thread ID of the thread for which the state is desired. The following special thread IDs can be used:

SCRIPTTHREADID_CURRENT	The currently executing thread.
SCRIPTTHREADID_BASE	The base thread (the thread in which the scripting engine was instantiated).

pstsState

Returned state of the indicated thread, as follows:

SCRIPTTHREADSTATE_NOTINSCRIPT	The specified thread is not currently servicing a scripted event, processing immediately executed script text, or running a script macro.
SCRIPTTHREADSTATE_RUNNING	The specified thread is actively servicing a scripted event, processing immediately executed script text, or running a script macro.

Returns

S_OK	Success.
E_POINTER	An invalid pointer was passed.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).

InterruptScriptThread

Interrupts the execution of a running script thread (either an event sink, an immediate execution, or a macro invocation). This method can be used to terminate a script that is stuck (for example, in an infinite loop). It may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

HRESULT IActiveScript::InterruptScriptThread(
[in] SCRIPTTHREADID *stidThread***,**
[in] const EXCEPINFO **pexcepinfo***,**
[in] DWORD *dwFlags*
);

Parameters

stidThread

Thread ID of thread to interrupt. The following special thread IDs can be used:

SCRIPTTHREADID_CURRENT	The currently executing thread.
SCRIPTTHREADID_BASE	The base thread (the thread in which the scripting engine was instantiated).
SCRIPTTHREADID_ALL	All threads. The interrupt is applied to all script methods currently in progress. Note that unless the caller has requested that the script be disconnected (with **SetScriptState()** to **SCRIPTSTATE_DISCONNECTED** or **_INITIALIZED)**, the next scripted event will cause script code to run again.

pexcepinfo

Error information associated with the error condition (for macro invocations and immediate execution, this is returned to the caller).

dwFlags

Option flags associated with the interruption.

SCRIPTINTERRUPT_DEBUG	If supported, enter the scripting engine's debugger at the current script execution point.
SCRIPTINTERRUPT_RAISEEXCEPTION	If supported by the scripting engine's language, let the script handle the exception. Otherwise, the script method is aborted and the error code is returned to the caller (the event source or macro invoker).

Returns

S_OK	Success, indicated thread will be interrupted.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).

Clone

Clones the current scripting engine (minus any current execution state), returning a loaded, unsited scripting engine in the current thread. The state of this new scripting engine should be identical to the state the original scripting engine would be in if it were transitioned back to the **initialized** state. Note that **Clone()** is an optimization of **IPersist*::Save()**, **CoCreateInstance()**, and **IPersist*::Load()**, so the state of the new scripting engine should be the same as if the state of the original scripting engine were saved and loaded into a new scripting engine. Named items will be duplicated in the cloned scripting engine, but specific object pointers for each item will be forgotten and will be obtained with **GetItemInfo()**. This allows an identical object model with per-thread entry points (an apartment model) to be used. This method is used for multithreaded server hosts that may run multiple instances of the same script. The scripting engine may return **E_NOTIMPL**, in which case the host can achieve the same result by duplicating the persistent state and creating a new instance of the scripting engine with **IPersist***. This method may be called from non-base threads without resulting in a non-base callout to host objects or to **IActiveScriptSite**.

HRESULT IActiveScript::Clone(
[out] IActiveScript ***ppscript*
);

Parameters

ppscript
The returned, unsited, cloned scripting engine. The host must create a site and call **SetScriptSite()** on the new scripting engine before it will be in the **initialized** state and, therefore, usable.

Returns

S_OK	Success.
E_NOTIMPL	Method not supported.
E_POINTER	An invalid pointer was passed.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).

IActiveScriptParse

If the ActiveX Scripting engine allows raw text code scriptlets to be added to the script, or allows expression text to be evaluated at runtime, it implements **IActiveScriptParse**. For interpreted scripting languages that have no independent authoring environment, such as Visual Basic Script, this provides an alternate mechanism (other than **IPersist**) to get script code into the scripting engine, and to attach script fragments to various object events.

```
Interface IActiveScriptParse : IUnknown
  {
  HRESULT InitNew();
  HRESULT AddScriptlet(
  [in]  LPCOLESTR  pstrDefaultName,
  [in]  LPCOLESTR  pstrCode,
  [in]  LPCOLESTR  pstrItemName,
  [in]  LPCOLESTR  pstrSubItemName,
  [in]  LPCOLESTR  pstrEventName,
  [in]  LPCOLESTR  pstrEndDelimiter,
  [in]  DWORD      dwFlags,
  [out] BSTR      *pbstrName,
  [out] EXCEPINFO *pexcepinfo);
  HRESULT ParseScriptText(
  [in]  LPCOLESTR  pstrCode,
  [in]  LPCOLESTR  pstrItemName,
  [in]  IUnknown  *punkContext,
  [in]  LPCOLESTR  pstrEndDelimiter,
  [in]  DWORD      dwFlags,
  [out] VARIANT   *pvarResult,
  [out] EXCEPINFO *pexcepinfo);
  };
```

InitNew

Before the scripting engine may be used, one of the following methods must be called: **IPersist*::Load()**, **IPersist*::InitNew()**, or **IActiveScriptParse:: InitNew()**. The semantics of this method are identical to **IPersistStreamInit::InitNew()**, in that this method tells the scripting engine to initialize itself. Note that it is not valid to call both **InitNew()** and **Load()**, nor is it valid to call **InitNew()** or **Load()** more than once.

HRESULT IActiveScriptParse::InitNew( );

Returns

S_OK	Success, the scripting engine has been initialized.
E_FAIL	An error occurred during initialization.

AddScriptlet

Adds a code scriptlet to the script. This method is used in environments where the persistent state of the script is intertwined with the host document and must be restored under the host's control, rather than though **IPersist**. The primary examples are HTML scripting languages that allow scriptlets of code imbedded in the HTML document to be attached to intrinsic events (for example, **ONCLICK="button1.text='Exit'"**).

HRESULT IActiveScriptParse::AddScriptlet(
[in] LPCOLESTR *pstrDefaultName*,
[in] LPCOLESTR *pstrCode*,
[in] LPCOLESTR *pstrItemName*,
[in] LPCOLESTR *pstrSubItemName*,
[in] LPCOLESTR *pstrEventName*,
[in] LPCOLESTR *pstrEndDelimiter* ,
[in] DWORD *dwFlags*,
[out] BSTR **pbstrName*,
[out] EXCEPINFO **pexcepinfo*
);

Parameters

pstrDefaultName
A default name to associate with the scriptlet. If the scriptlet does not contain naming information (as in the **ONCLICK** example above), this name will be used to identify the scriptlet. If this parameter is **NULL**, the scripting engine will manufacture a unique name if necessary.

pstrCode
The scriptlet, in textual form, to add. The interpretation of this string is language dependent.

pstrItemName
Identifies the named item this scriptlet is associated with. This parameter, in addition to **pstrSubItemName**, identifies the specific object for which the scriptlet is an event handler.

pstrSubItemName
Identifies a sub-object of the named item with which this scriptlet is associated; this name must be found in the named item's type information. A **NULL** is passed if the scriptlet is to be associated with the named item instead of a sub-item. This parameter, in addition to **pstrItemName**, identifies the specific object for which the scriptlet is an event handler.

pstrEventName
Identifies the event for which the scriptlet is an event handler.

pstrEndDelimiter
When **pstrCode** is parsed from a stream of text, the host typically uses a delimiter, such as two single quotation marks (''), to detect the end of the scriptlet. The delimiter the host used is passed in here, so that the scripting engine can provide some conditional primitive preprocessing (for example, replacing a single quotation mark ['] with two single quotation marks for use as a delimiter). Exactly how (and if) the scripting engine makes use of this information is left to the scripting engine. NULL may be passed, indicating that the host did not use a delimiter to detect the end of the scriptlet.

dwFlags
Flags associated with the scriptlet:

SCRIPTTEXT_ISVISIBLE	Indicates that the script text should be visible (and, therefore, callable by name) as a global method in the namespace of the

script.

SCRIPTTEXT_ISPERSISTENT — Indicates that the code added during this call should be saved if the scripting engine is saved, such as via **IPersist*::Save()**, or if the scripting engine is reset via a transition back to the **initialized** state. Indicates that the script text should be visible (and, therefore, callable by name) as a global method in the namespace of the script.

pbstrName

The actual name used to identify the scriptlet. This will be, in order of preference: a name explicitly specified in the scriptlet text, the default name provided in **pstrDefaultName**, or a unique name synthesized by the scripting engine.

pexcepinfo

Pointer to a structure containing exception information. This structure should be filled in if **DISP_E_EXCEPTION** is returned.

Returns

S_OK	Success, the scriptlet has been added to the script, and its name is returned in **pbstrName**.
OLESCRIPT_E_INVALIDNAME	The default name supplied is invalid in this scripting language.
OLESCRIPT_E_SYNTAX	There was a unspecified syntax error in the scriptlet.
DISP_E_EXCEPTION	An exception occurred in the parsing of the scriptlet. Exception information is returned in **pexcepinfo**.
E_UNEXPECTED	The call was not expected (for example, the scripting engine has not yet been loaded or initialized).
E_NOTIMPL	Method is not supported, the scripting engine does not support adding event-sinking scriptlets.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.

ParseScriptText

Parses the given code scriptlet, adding declarations into the name space and evaluating code as appropriate. If the scripting engine is in the **initialized** state, no code will actually be evaluated during this call, rather, such code is queued and executed when the scripting engine is transitioned into (or through) the **started** state. Because execution is not allowed in the **initialized** state, it is an error to call this method with the **SCRIPTTEXT_ISEXPRESSION** flag when in the **initialized** state. The scriptlet may be an expression, a list of statements, or anything allowed by the script language. For example, this method is used in the evaluation of the HTML <SCRIPT> tag, which allows statements to be executed as the HTML page is being constructed, rather than just compiling them into the script state. Note that the code passed to this method must be a valid, complete portion of code. For example, in VBScript it is illegal to call this method once with **Sub Foo(x)** and then a second time with **End Sub**. The parser must not wait for the second call to complete the subroutine, but rather must generate a parse error because a subroutine declaration was started but not completed.

HRESULT IActiveScriptParse::ParseScriptText(
[in] LPCOLESTR *pstrCode*,
[in] LPCOLESTR *pstrItemName*,
[in] IUnknown **punkContext*,
[in] LPCOLESTR *pstrEndDelimiter*,
[in] DWORD *dwFlags*,
[out] VARIANT **pvarResult*,
[out] EXCEPINFO **pexcepinfo*
);

Parameters

pstrCode
The scriptlet, in textual form, to evaluate. The interpretation of this string is language dependent.

pstrItemName
Identifies the context in which the scriptlet is to be evaluated. If **NULL**, then the code is evaluated in the scripting engine's global context.

punkContext
Reserved for use in a debugging environment, where such a context may be provided by the debugger to represent an active runtime context. If **NULL**, **pstrItemName** is used to identify the context.

pstrEndDelimiter
When **pstrCode** is parsed from a stream of text, the host typically uses a delimiter, such as two single quotation marks ("), to detect the end of the scriptlet. The delimiter the host used is passed in here, so that the scripting engine can provide some conditional primitive preprocessing (for example, replacing a single quotation mark ['] with two single quotation marks for use as a delimiter). Exactly how (and if) the scripting engine makes use of this information is left to the scripting engine. NULL may be passed, indicating that the host did not use a delimiter to detect the end of the scriptlet.

dwFlags
Flags associated with the scriptlet:

SCRIPTTEXT_ISEXPRESSION — If the distinction between a computational expression and a statement is important but syntactically ambiguous in the script language, this flag specifies that the scriptlet is to be interpreted as an expression, rather than as a statement or list of statements. By

	default, statements are assumed unless the correct choice can be determined from the syntax of the scriptlet text.
SCRIPTTEXT_ISPERSISTENT	Indicates that the code added during this call should be saved if the scripting engine is saved, such as via **IPersist*::Save()**, or if the scripting engine is reset via a transition back to the **initialized** state.
SCRIPTTEXT_ISVISIBLE	Indicates that the script text should be visible (and, therefore, callable by name) as a global method in the namespace of the script.

pvarResult

Pointer to where the result is to be stored, or NULL if the caller expects no result (**SCRIPTEXT_ISEXPRESSION** is not set).

pexcepinfo

Pointer to a structure containing exception information. This structure should be filled in if **DISP_E_EXCEPTION** is returned.

Returns

S_OK	Success, the expression or statement(s) have been evaluated. The result, if any, is returned in **pvarResult**.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.
E_UNEXPECTED	The call was not expected (for example, the scripting engine is in the **uninitialized** or **closed** state, or the **SCRIPTEXT_ISEXPRESSION** flag was set and the scripting engine is in the **initialized** state).
DISP_E_EXCEPTION	An exception occurred in the processing of the scriptlet. Exception information is returned in **pexcepinfo**.
OLESCRIPT_E_SYNTAX	There was a unspecified syntax error in the scriptlet.
E_NOTIMPL	The function is not supported. The scripting engine does not support run-time evaluation of expressions or statements.

IActiveScriptError

An object implementing this interface is passed to **IActiveScriptSite::OnScriptError()** whenever the scripting engine encounters an unhandled error. The host then calls methods on this object to obtain information about the error that ocurred.

```
Interface IActiveScriptError : IUnknown
  {
  HRESULT GetExceptionInfo([out] EXCEPINFO *pexcepinfo);
  HRESULT GetSourcePosition(
    [out] DWORD *pdwSourceContext,
    [out] ULONG *pulLineNumber,
    [out] LONG  *pichCharPosition);
  HRESULT GetSourceLineText([out] BSTR *pbstrSourceLine);
  };
```

GetExceptionInfo

Retrieves error information.

HRESULT GetExceptionInfo(
[out] EXCEPINFO **pexcepinfo*
);

Parameters

pexcepinfo
Points to an EXCEPINFO structure that contains error information.

Returns

S_OK	Success.
E_FAIL	An error occurred.

GetSourcePosition

Retrieves the error location in the source code.

HRESULT GetSourcePosition(
[out] DWORD **pdwSourceContext*,
[out] ULONG **pulLineNumber*,
[out] LONG **pichCharPosition*
);

Parameters

pdwSourceContext
Points to a cookie that identifies the context. (This interpretation of this parameter depends on the host application.)

pulLineNUmber
Points to a value that specifies the line number in the source file where the error occurred..

pichCharPosition
Points to a value that specifies the character position in the line where the error occurred..

Returns

S_OK	Success.
E_FAIL	An error occurred.

GetSourceLineText

Retrieves the line in the sourcefile where an error occurred.

HRESULT GetSourceLineText(
[out] BSTR **pbstrSourceLine*
);

Parameters

pbstrSourceLine
Points to the line of source code in which the error occurred.

Returns

S_OK	Success.
E_FAIL	An error occurred.

The ActiveX Scripting Site Object

The host must create a site for the ActiveX Scripting engine by implementing **IActiveScriptSite**. Usually this site will be associated with the container of all the objects that are visible to the script (for example, the OLE Controls). Typically, this container will correspond to the document or page being viewed. The Internet Explorer (IE), for example, would create such a container for each HTML page being displayed. Each OLE Control (or other OLE Automation object) on the page, and the scripting engine itself, would be enumerable within this container.

IActiveScriptSite

```
interface IActiveScriptSite : IUnknown
  {
  HRESULT GetLCID([out] LCID *plcid);
  HRESULT GetItemInfo(
      [in]  LPCOLESTR   pstrName,
      [in]  DWORD       dwReturnMask,
      [out] IUnknown  **ppunkItem,
      [out] ITypeInfo **ppTypeInfo);
  HRESULT GetDocVersionString([out] BSTR *pstrVersionString);
  HRESULT OnScriptTerminate([in] VARIANT *pvarResult, [in] EXCEPINFO
*pexcepinfo);
  HRESULT OnStateChange([in] SCRIPTSTATE ssScriptState);
  HRESULT OnScriptError([in] IActiveScriptError *pase);
  HRESUT    OnEnterScript();
  HRESULT   OnLeaveScript();
  };
```

GetLCID

Retrieves the internationalization locale ID associated for the host's UI. This allows error strings and other UI surfaced by the scripting engine to appear in the appropriate language for the end user. If this method returns **E_NOTIMPL**, the system **LCID** should be used.

HRESULT IActiveScriptSite::GetLCID(
[out] LCID **plcid*
);

Parameters

plcid
The returned language ID for UI displayed by the scripting engine.

Returns

S_OK	Success, **lcid** contains the appropriate **LCID**.
E_POINTER	An invalid pointer was passed.
E_NOTIMPL	Use the sytem **LCID**.

GetItemInfo

Allows the scripting engine to obtain information about an item added with **IActiveScript::AddNamedItem()**. Note that if the associated bit for each out parameter pointer is not set in dwReturnMask, the respective result is not returned. This improves performance, for example, in the case where an **ITypeInfo** is not needed for an item.

HRESULT IActiveScriptSite::GetItemInfo(
[in] LPCOLESTR *pstrName*,
[in] DWORD *dwReturnMask*,
[out] IUnknown ***ppunkItem*,
[out] ITypeInfo ***ppTypeInfo*
);

Parameters

pstrName
The name associated with the item, as specified in **IActiveScript::AddNamedItem()**.

dwReturnMask
A bit mask specifying what information about the item should be returned. The scripting engine should take care to request the minimum needed information because some of the return parameters (for example, **ITypeInfo**) may take considerable time to load or generate.

SCRIPTINFO_IUNKNOWN	Return the **IUnknown** for this item.
SCRIPTINFO_ITYPEINFO	Return the **ITypeInfo** for this item.

ppunkItem
The **IUnknown** associated with the item. The scripting engine can **QueryInterface()** for **IDispatch** through this. **NULL** is returned if **SCRIPTINFO_IUNKNOWN** is not set in **dwReturnMask**. A **NULL** is also returned when there is no object to be associated with the name; this mechanism is used to create a simple class when the named item was added with the **SCRIPTITEM_CODEONLY** flag set.

ppTypeInfo
The **ITypeInfo** associated with the item. **NULL** is returned if **SCRIPTINFO_ITYPEINFO** is not set in **dwReturnMask**. **NULL** is also returned if type information is not available for this item. In this case, the object cannot source events, and name binding must be realized with **IDispatch::GetIDsOfNames()**. Note that this **ITypeInfo** describes the coclass (**TKIND_COCLASS**), since the object may support multiple interfaces and event interfaces. If the item supports **IProvideMultipleTypeInfo**, then this **ITypeInfo** corresponds to the **ITypeInfo** of index zero obtained from **IProvideMultipleTypeInfo::GetInfoOfIndex()**.

Returns

S_OK	Success, the relevant out parameters have been set.
E_POINTER	An invalid pointer was passed.
E_INVALIDARG	An invalid argument was passed.
TYPE_E_ELEMENTNOTFOUND	An item of the specified name was not found.

GetDocVersionString

Returns a host-defined string that uniquely identifies the current document version from the host's point of view. If the related document has changed outside the scope of ActiveX Scripting (as in the case of an HTML page being edited with NotePad), the scripting engine may choose to save this along with its persisted state, forcing a recompile the next time the script is loaded.

HRESULT IActiveScriptSite::GetDocVersionString(
[out] BSTR **pbstrVersionString*
);

Parameters

pstrVersionString
The host-defined document version string.

Returns

S_OK	Success, the document version string is returned in **pstrVersionString**.
E_NOTIMPL	The method is not supported. The scripting engine should assume that the script is in synch with the document.

OnScriptTerminate

Informs the host that the script has completed execution. This method is called before **OnStateChange(SCRIPTSTATE_INITIALIZED)** is completed. It can be used to return completion status/results to the host. Note that many script languages, which are based on sinking events from the host, have life spans that are defined by the host. In this case, this method may never be called.

HRESULT IActiveScriptSite::OnScriptTerminate(
[in] VARIANT **pvarResult*,
[in] EXCEPINFO **pexcepinfo*
);

Parameters

pvarResult
Pointer to where the script result is stored, or **NULL** if the script produced no result.

pexcepinfo
Pointer to a structure containing exception information generated at script termination, or **NULL** if no exception was generated.

Returns

S_OK	Success.

OnStateChange

Informs the host that the scripting engine has changed states.

HRESULT IActiveScriptSite::OnStateChange(
[in] SCRIPTSTATE *ssScriptState*
);

Parameters

ssScriptState
The new script state. See **IActiveScript::GetScriptState()** for a description of the states.

Returns

S_OK	Success.

OnScriptError

This method is called when an execution error occurs while running the script.

HRESULT IActiveScriptSite::OnScriptError(
[in] *IActiveScriptError*pase*
);

Parameters

pase
An object from which error information associated with the error condition can be obtained.

Returns

S_OK	The scripting engine should continue running the script as best as possible (perhaps abandoning the processing of this event).
S_FALSE	The scripting engine should continue running the script in the debugger, if a debugger is available. If a debugger is not available, then this should be treated just like **E_FAIL**.
E_FAIL	The scripting engine should abort execution of the script and return it to a **initialized** state. In this case, the **pexcepinfo** obtained from **IActiveScriptError::GetExceptionInfo()** is generally passed on to **OnScriptTerminate()**.

OnEnterScript

Called whenever the scripting engine begins to execute script code. This must be called by the scripting engine on every entry or re-entry into the scripting engine. For example, if the script calls an object that then fires an event handled by the scripting engine, the scripting engine must call **OnEnterScript()** before executing the event, and must call **OnLeaveScript()** after executing the event before returning to the object that fired the event. Calls can nest. For every **OnEnterScript()** there must eventually be an **OnLeaveScript()**.

HRESULT IActiveScriptSite::OnEnterScript( );

OnLeaveScript

Called whenever the scripting engine returns from executing script code. This must be called by the scripting engine before returning control to a caller that entered the scripting engine. For example, if the script calls an object that then fires an event handled by the scripting engine, the scripting engine must call **OnEnterScript()** before executing the event, and must call **OnLeaveScript()** after executing the event before returning to the object that fired the event. Calls can nest. For every **OnEnterScript()** there must eventually be an **OnLeaveScript()**.

HRESULT IActiveScriptSite::OnLeaveScript( );

IActiveScriptSiteWindow

This interface is implemented by hosts that support UI on the same object as **IActiveScriptSite**. Hosts that do not support UI, such as servers, would not implement this interface. The scripting engine accesses this interface by a **QueryInterface()** from **IActiveScriptSite**.

```
interface IActiveScriptSiteWindow : IUnknown
  {
  HRESULT GetWindow([out] HWND *phwnd);
  HRESULT EnableModeless([in] BOOL fEnable);
  };
```

GetWindow

Returns a window that can act as the owner of a pop-up window the scripting engine wishes to display. This method is similar to **IOleWindow::GetWindow()**.

HRESULT IActiveScriptSite::GetWindow(
[out] HWND **phwnd*
);

Parameters

phwnd
Specifies a window that can act as the owner window for pop-ups.

Returns

S_OK	Success.
E_FAIL	An error occurred.

EnableModeless

Causes the host to enable or disable its main window as well as any modeless dialog boxes. This method is identical to **IOleInPlaceFrame::EnableModeless()**.

HRESULT IActiveScriptSite::EnableModeless(
[in] BOOL *fEnable*
);

Parameters

fEnable
Specifies whether the modeless dialogs are to be enabled (TRUE) or disabled (FALSE). Calls nest.

Returns

S_OK	Success.
E_FAIL	An error occurred.

(*) Certain interpreted script languages (for example, VBScript) running in specific host environments (for example, Internet Explorer) may rarely (or never) be called upon to save or restore a script state via **IPersist***. Instead, **IActiveScriptParse** is used by calling **IActiveScriptParse::InitNew()** to create a blank script, then scriptlets are added and connected to events with **IActiveScriptParse::AddScriptlet()** and general code is added via **IActiveScriptParse::ParseScriptText()**. Nonetheless, a scripting engine should fully implement at least one **IPersist*** scheme (preferably **IPersistStreamInit**), as other host applications may try to make use of them.

(**) Unfortunately, this also means that the scripting engine must be implemented as an in-process server, since OLE does not currently support interprocess marshaling of free-threaded objects.

(***)For more information, see *A Word About Threading*.

Palette Management for ActiveX Objects

Note: This document is an early release of the final specification. It is meant to specify and accompany software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of the final specification or software. Microsoft assumes no responsibility for any damages that might occur either directly or indirectly from these inaccuracies. Microsoft may have trademarks, copyrights, patents or pending patent applications, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you a license to these trademarks, copyrights, patents, or other intellectual property rights.

Introduction

This document describes palette management for ActiveX Objects, specifically ActiveX Controls and ActiveX Document Objects. The palette management rules below allow multiple controls inside a form or in an HTML page to display correctly while still leaving enough flexibility to allow a control to demand palette control. In general, containers are responsible for palette management, and contained objects (e.g. controls) should only realize their palettes in the background.

Palette Management for ActiveX Controls and Containers

This section describes how ActiveX controls and containers should manage the palette.

Palette Selection

Because there may be more than one control on a page or form, it is up to the container to choose a common palette, and every control should realize its paleete in the background. Anything else would result in chaos as each control realized its palette while painting. The display would flicker madly and most controls would look awful.

How the container chooses the common palette is up to the implementor of the container. Some containers may use the UI active object's palette, others may compute the palette at runtime. Microsoft Internet Explorer 3.0 will either use a default halftone palette, or use the UI active object's palette. This policy may change in future versions of Microsoft Internet Explorer.

Ambient palette property

The container should make its palette available as an ambient property: DISPID_AMBIENT_PALETTE. This property is defined in OLE Controls documentation and will be implemented by Microsoft Internet Explorer 3.0 and other Microsoft OLE containers. Controls can receive notification of changes to this property by implementing: IOleControl::OnAmbientPropertyChange()

Containers should realize the ambient palette in the foreground before calling this function or IViewObject2::Draw().

Older containers that do not implement this property will typically iterate through their controls, forwarding the WM_QUERYNEWPALETTE message until a control returns TRUE (indicating it has realized a palette).

Newer containers that implement DISPID_AMBIENT_PALETTE should never send WM_QUERYNEWPALETTE to their controls.

Palette change notification

Palette aware controls that wish to be notified of changes to the ambient palette should implement IOleControl::OnAmbientPropertyChange() and handle DISPID_AMBIENT_PALETTE and DISPID_UNKNOWN. The DISPID_UNKNOWN change can be sent when more than one property changes. When this happens, controls should explicitly check for a changed palette.

Existing containers that do not define an ambient palette will send WM_PALETTECHANGED messages. Controls are encouraged to handle this message as well.

Note: controls can still realize a different palette (in the background). The ambient palette property is only useful if a control wishes to optimize its display to the palette of the container. If this is not necessary, the control should ignore the ambient palette property.

IViewObject2::GetColorSet()

All palette aware controls should implement IViewObject2::GetColorSet() if they wish to specify a palette preference. Containers can use the return value to determine if a control is palette aware. Containers may also use the color information returned to choose what palette to realize.

E_NOTIMPL	The control is not palette aware
S_FALSE	The control is palette aware but does not have a palette at this time.
S_OK	The control is palette aware and has returned its palette to the container.

It should be noted that the container does not have to call IViewObject2::GetColorSet(). The container may determine it's palette independently of its controls.

Realizing the palette

Current OLE guidelines allow the UI active object to realize its own palette in the foreground. Under the new guidelines, this is no longer the case.

Controls should only realize their palette in the foreground when receiving WM_QUERYNEWPALETTE message. In every other situation, the **control must realize its palette in the background.** Containers that implement DISPID_AMBIENT_PALETTE will never send this message.

Summary

- Containers determine the palette
- Containers may choose to implement DISPID_AMBIENT_PALETTE
- Containers must realize the ambient palette before calling IViewObject2::Draw() or IOleControl::OnAmbientPropertyChange()
- Controls must **always** realize their palette in the background. The only exception is when receiving the WM_QUERYNEWPALETTE message.
- Controls should implement IViewObject2::GetColorSet() if they care to tell the container about their palette preferences.
- Controls should implement IOleControl::OnAmbientPropertyChange() if they wish to be notified about changes in the ambient palette property in order to optimize their display to the container's palette.

Palette Management for ActiveX Document Objects

The following sections describe issues related to palette management by OLE Document Objects. In general, the palette management scheme for document objects is the same as the scheme used for controls, except that document objects do not receive ambient properties from their container.

The Responsibilities of the Document Object

Palette management for document objects is similar to palette management for controls. The following are the responsibilities of the document object:

- Document objects should implement IViewObject2::GetColorSet if interested in notifying the container about the palette they wish to display with. This is exactly the same as the corresponding responsibility for Controls (above).
- Document objects should realize their palette in the background unless when handling WM_QUERYNEWPALETTE messages (forwarded from the container). Again, this is exactly the same as the corresponding responsibility for Controls (above).

The only difference between the responsibilities of controls and document objects is that document objects do not handle the DISPID_AMBIENT_PALETTE property.

Single Documnet Case

In the most common cases, a DocObject container will only activate one DocObject at a time. In such a situation, it is acceptable that the activated DocObject should have complete control over the palette which includes managing palette issues for any controls or content within that document.

The DocObject container in this case has nothing to do with palettes and need not be concerned with palette issues. It should leave palette handling up to the DocObject - in other words, it should forward all Windows palette management messages on to the DocObject.

Example code for the window procedure of the Document Object host:

```
    case WM_PALETTECHANGED:
    case WM_QUERYNEWPALETTE:
    {
        HWND hwnd;
        IOleWindow *pOleWindow;
        if (pDocObj && SUCCEEDED(pDocObj->QueryInterface(IID_IOleWindow,
(LPVOID*)&pOleWindow))) {
            LRESULT lres = 0;
            if (SUCCEEDED(pOleWindow->GetWindow(&hwnd))) {
                lres = SendMessage(hwnd, uMsg, wParam, lParam);
            }
            pOleWindow->Release();
            return lres;
        }
        break;
    }
```

Multiple DocObject Case

In rare cases, a DocObject container may be able to activate more than one DocObject at a time within multiple container frames. While this user interface is discouraged at this time, it is still possible to achieve.

However, no palette-management solution exists for this scenario because there is currently no protocol for communicating palettes between a DocObject and its container. Therefore the container cannot create a palette suitable to all DocObjects that it has activated.

Because of this, the activated DocObject in the foreground has control over the palette and should use foreground palette rendering. Other activated DocObjects in the background use background palette rendering. The DocObject container itself does not participate in palette management at all.

Introduction

This document describes how to support Platform for Internet Content Selection (PICS) ratings on the Windows platform. It is intended for Internet Content Providers (ICPs) and programmers who are developing web authoring tools. PICS is a W3C group chartered to define a standard for the creation of rating systems and ratings information. Microsoft views PICS as the best direction for solving the problems with minors accessing adult content on the Internet.

The PICS Standard

This section describes how the PICS system works, shows an example of rating information within an HTML page, and briefly describes the PICS specification.

How the PICS System Works

The PICS system has two types of components: rating systems and rating labels. A *rating system* defines the criteria for how content is rated. PICS has created a metalanguage for defining the different categories and the different values for rating content. For example, the Motion Picture Association of America (MPAA) rating system has one category with five values: G, PG, PG-13, R, NC-17. Using the PICS specification, rating systems with multiple categories can be defined.

The *rating label* is the actual rating information. The label can appear on a web page as part of the HTML content, on a third party web site, or anywhere else that a particular client application "knows" to look for them. Each item of content that is rated would have it's own label. PICS has defined the syntax for these rating labels, as well as extensions to many Internet protocols to support labels.

The PICS system works by matching labels to their associated systems. The client software, either part of the web browser or a separate application, allows the parent to decide what levels of content they want to allow their children to see for each rating system. Then, as the child uses the PICS aware web browser or application, the web browser finds the rating labels for the content and checks against the values the parent has set by calling a function in the Internet rating API. If the content is rated higher than the child is allowed, the content is not displayed.

Example of HTML Tagged Ratings

Ratings are specified by using an additional tag within the HTML header (<HEAD> rating information </HEAD>). The rating is in English text, as defined by the PICS specification. More details are provided in the full PICS documentation found on the PICS web site.

The following examples shows the general format of rating information within an HTML page:

```
<HTML>
<HEAD>
<META http-equiv="PICS-Label" content='(PICS-1.0 "http://www.rsac.org"
labels on "1996.11.05T08:15-0500"
until "1996.12.31T23:59-0000"
for "http://www.rsac.org/index.html"
by "RSAC "
ratings (language 2 nudity 0 violence 1))'>
</HEAD>
```

The core of the rating label is the *rating* string. This provides a list of category/value pairs. For each category, the name and the associated value are listed.

What PICS Has Defined

PICS has defined two specifications, one for creating rating systems, and one for creating rating labels. A rating system defines the criteria for how content is rated. Based on the PICS specification for rating systems, any group or individual could define a system for rating content. Using the PICS specification, you could take the MPAA rating system and define it as a PICS compliant rating system.

The PICS specification is quite extensive, and allows for a system to contain multiple categories. The MPAA system has only one category with five values in it: G, PG, PG-13, R, NC-17. It has been criticized for being too simplistic and not offering granularity on multiple dimensions. PICS solves this problem by allowing the creator of the rating system to have great flexibility in how the system is defined.

PICS has not defined an actual rating system or an actual scheme for rating Internet content. PICS is a framework and a language that allows any group or organization to rate content on their own criteria. PICS has not created a replacement for the movie rating system, nor does it intend to.

For more information abouut rating systems, see the PICS specification on their web site: *http://www.w3.org*.

Amendments to the PICS specification

The PICS specification makes no provisions for storing information on rating systems locally. On the Windows platform, the text description for PICS rating systems is contained a text file with an .RAT extension. This file would contain all of the category and value information that PICS has defined as necessary. Microsoft may define fields for information about a given rating system, including it's PICS rating server URL if it has one.

What Internet Content Providers Should Know About PICS

Internet Content Providers (ICPs) need to be aware that they must take the initiative in getting their content rated by a PICS compliant rating system. For a rating system to be useful, the browser application must deny access to sites that are unrated. ICPs should contact the PICS committee for a listing of ratings groups that are based on PICS.

Microsoft is committed to supporting the PICS standard and providing API and reference information for other products that wish to support PICS. In addition, Microsoft is assisting several third party groups to create PICS based rating systems. In particular, the Recreation Software Advisory Council (RSAC) has created a rating system called RSACi. Their web site, *http://www.rsac.org*, provides a simple online mechanism for getting content rated through their system.

What Developers of Web Authoring Tools Should Know About PICS

Developers of web authoring tools need to be PICS aware, in that they support the ability to easily enter META tags into web pages. Web authoring tools can help promote the PICS technology by providing templates or wizards for adding PICS ratings to documents.

Of significant importance is the ability to maintain ratings for groups or "virtual trees" of pages. Some PICS compliant rating systems may offer ratings for groups of content. This will allow larger sites to section adult content from general audience content. Web Authoring and management tools need to provide the ability to propagate ratings information across user-defined groups of pages.

Microsoft intends to provide support for the easy addition of PICS compliant ratings information into appropriate Internet content tools.

Internet Ratings API

The Microsoft® Internet Ratings API provides Win32-based applications with easy access to support PICS based Internet ratings and related services. These functions provide any application with access to a PICS rating parsing engine, as well as a mechanism for comparing ratings against the Nashville user information settings. This API will be available as part of Nashville (Internet Add-On Pack).

The functions in the Internet Ratings API set facilitate access to the ratings support in the following ways:

- *Provides a common interface for parsing and obtaining PICS compliant rating information.* A browser or other application can support ratings with the addition of only a few function calls. By providing task oriented functions, applications need only make a few function calls to support ratings.
- *Eliminates the need for individual applications to define their own control systems.* With this API, applications need only to find the rating information for a given piece of content. Additional APIs for retrieving some types of Internet ratings are provided.

Overview of Internet Ratings Functions

Note The following overview should be considered a prerelease draft of the API specification.

The PICS specification allows for ratings information to appear in HTML tags, HTTP headers, and on a third party URL. For performance reasons, each individual browser is responsible for obtaining HTML and HTTP based ratings information (For details on obtaining these mechanisms, consult the PICS specification).

The following table summarizes the PICS support API functions. This table lists all of the functions for finding, pairing, and comparing rating information.

Function	Description
RatingObtainQuery	Obtains ratings from certain locations and compares them.
RatingObtainCancel	Cancels a call to RatingObtainQuery.
RatingCheckUserAccess	Takes in a PICS rating, parses it, and compares it against what a specified user can see.
RatingAccessDeniedDialog	Displays a system dialog informing the user that access has been denied.
RatingFreeDetails	Frees a pointer to denial information.
RatingGetSupervisorOverride	Requests override by a supervisor user.
RatingIsUserSupervisor	Returns Boolean value for whether a given user is a supervisor (Nashville only).
RatingEnabledQuery	Specifies whether ratings are on or off.

Using the Internet Ratings Functions

A browser or other application would typically use the ratings functions in the following fashion:

- The browser starts and the user connects to the Internet.
- The user enters a URL.
- The browser calls the RatingEnabledQuery function. If the function returns S_FALSE, the browser allows access to the content.
- The browser calls the RatingObtainQuery function and passes the URL. The function checks the local list and then the third party list, or any third party add-on, if any are activated. If the function finds a rating for the URL, it calls RatingCheckUserAccess with the rating. If RatingObtainQuery does not return an error immediately, eventually the browser's callback function will be called.
- Meanwhile, the browser searches for a rating label within the content being downloaded from the site. If it finds an HTML/HTTP rating, it calls the RatingCheckUserAccess function.
- If no rating label is found within the document, the browser calls the RatingCheckUserAccess function with a NULL rating label, to determine whether the user may access unrated content.
- When both RatingCheckUserAccess and the RatingObtainQuery callback functions have been called, the browser has received at least one explicit access-allowed or -denied report. If RatingObtainQuery explicitly allowed or denied access, that result should override any rating found in the document and the user's ability to see unrated content.
- If access is denied, the browser calls the RatingAccessDeniedDialog function. Depending on the supervisor's response to the dialog box, this function can indicate that access should be allowed after all.
- If access is allowed, the browser displays the content.

Finding HTML/HTTP Rating Labels

The PICS specification defines the BNF syntax for ratings information, or ratings nuggets. PICS specifies that rating nuggets can appear as part of HTML documents, within the Header tags. The PICS specification also allows for ratings information to appear as part of the HTTP headers.

For performance and logistic reasons, it is the responsibility of the browser to obtain HTML and HTTP level ratings. Details on how to find HTML-based ratings are specified as part of the PICS specification. As of 2/28/96, HTML based ratings appears to be the most common location for ratings information.

Support for HTTP ratings may eventually be provided via URL_Moniker, part of the ActiveX SDK. Details on HTTP ratings, and the associated HTTP header are specified by PICS.

The PICS committee may eventually provide reference code for applications that wish to find HTML/HTTP based ratings.

Reference

This section contains detailed reference information about the functions in the Microsoft Internet Ratings API.

RatingAccessDeniedDialog

Displays a modal dialog box telling the user that access to the requested content is denied. The supervisor can set an option that adds controls to the dialog box which allow a supervisor to enter a username and password to override the rating and allow access.

HRESULT RatingAccessDeniedDialog(
HWND *hwndParent*,
LPCTSTR *pszUsername*,
LPCTSTR *pszContentDescription*,
LPVOID *pRatingDetails* **);**

Parameters

hwndParent
Handle of the parent window for the modal dialog box.

pszUsername
Address of a string that contains the username of the user who was denied access. If this parameter is NULL, the username is extracted from the structure specified by *pRatingDetails*, or the current username is used.

pRatingDetails
Opaque pointer returned by the RatingCheckUserAccess function. This parameter can provide the dialog box with additional information about why access was denied, such as whether the site is considered rated, which ratings exceeded limits, and where the rating was obtained.

Return Values

Returns one of these values:

S_OK	The supervisor entered their password. Access should be allowed.
S_FALSE	The user is still denied access to the content.
Error values	An error occurred.

See Also

RatingCheckUserAccess

RatingFreeDetails

Frees the detail information returned by the RatingCheckUserAccess function.

HRESULT RatingFreeDetails(
LPVOID *pRatingDetails* **);**

Parameters

pRatingDetails
Opaque pointer returned by the RatingCheckUserAccess function.

See Also

RatingCheckUserAccess

RatingCheckUserAccess

Determines whether the user is allowed to view the given content based on the associated rating label. An application calls this function when it finds a rating label.

HRESULT RatingCheckUserAccess(
LPCTSTR *pszUsername*,
LPCTSTR *pszURL*,
LPCTSTR *pszRatingInfo*,
LPBYTE *pData*,
DWORD *cbData*,
LPVOID **ppRatingDetails* **);**

Parameters

pszUsername
Address of a string that contains the name of the user whose access to the content should be validated. A NULL value means the currently logged on user. For Internet Explorer version 3.0, NULL is the only valid value for this parameter.

pszURL
Address of the URL to which the rating label is to apply. The function uses parameter validate the corresponding field in the rating label, to verify that the rating label actually belongs to the URL for which label was obtained. This parameter can be NULL if a URL is not applicable.

pszRatingInfo
Address of the PICS label to parse and verify (for example, the text between the "content" single quotes in the example in Example of HTML Tagged Ratings). This parameter can be NULL if the application cannot find a rating in the content itself; in this case, the function returns a value that indicates whether the user is allowed to see unrated content.

pData and *cbData*
Address of the content being rated and the length, in bytes, of the content data stream. These parameters may be necessary if the rating itself contains a hash of the data for security reasons. If *pData* is NULL, the *cbData* parameter is ignored, and the hash cannot be validated.

ppRatingDetails
Address of a variable that contains a pointer to an opaque data structure that describes in more detail why access to the content is denied. This parameter has meaning only to the RatingAccessDeniedDialog function. This parameter can be NULL, in which case the dialog box created by RatingAccessDeniedDialog cannot report detailed information to the user. If this parameter is not NULL, the application it must call the RatingFreeDetails function to free the data, no matter what value the **RatingCheckUserAccess** function returns or what value is stored in *ppRatingDetails*.

Return Values

Returns one of these values:

S_OK	Access is allowed.
S_FALSE	Access is denied.
Error values	Content should be considered unrated.

Remarks

This function allows ratings of any content type, including CD-ROM's, newsgroups, Web sites, and so on, provided the client application can find the rating information. The RatingObtainQuery function

makes an internal call to the **RatingCheckUserAccess** function before calling the application's callback function.

See Also

RatingAccessDeniedDialog, RatingFreeDetails, RatingObtainQuery

RatingEnabledQuery

Reports whether ratings should be enforced or not. If the supervisor has not configured any restrictions or has temporarily disabled them, the application may allow access to all content and need not call any other ratings functions.

HRESULT RatingEnabledQuery(void)

Return Values

Returns one of these value:

S_OK	Ratings are enabled and should be enforced.
S_FALSE	Ratings are disabled and should be ignored.

RatingGetSupervisorOverride

Displays a modal dialog box telling the user that access to the requested content is denied. This function is similar to RatingAccessDeniedDialog, except the dialog box displayed by this function always contains controls for entering the supervisor's username and password. An application can use this function to override the global setting that controls the presence of this user interface element.

HRESULT RatingGetSupervisorOverride(
HWND *hwndParent*,
LPCTSTR *pszUsername*,
LPCTSTR *pszContentDescription*,
LPVOID *pRatingDetails* **);**

Parameters

hwndParent
Handle of the parent window for the modal dialog box.

pszUsername
Address of a string that contains the username of the user who was denied access. If this parameter is NULL, the username is extracted from the structure specified by *pRatingDetails*, or the current username is used.

pRatingDetails
Opaque pointer returned by the RatingCheckUserAccess function. This parameter can provide the dialog box with additional information about why access was denied, such as whether the site is considered rated, which ratings exceeded limits, and where the rating was obtained.

Return Values

Returns one of these values:

S_OK	The supervisor entered their password. Access should be allowed.
S_FALSE	The user is still denied access to the content.
Error values	An error occurred.

See Also

RatingAccessDeniedDialog, RatingCheckUserAccess

RatingIsUserSupervisor

Reports whether the specified user is a supervisor. This function is available on Nashville only; in Internet Explorer 3.0, anyone who knows the global supervisor password is a supervisor.

HRESULT RatingIsUserSupervisor(
LPCTSTR *pszUsername* **);**

Parameters

pszUsername
Address of a string that contains the name of the user. If this parameter is NULL, the function uses the name of the user who is currently logged on.

Return Values

Returns one of these values:

S_OK	User is a supervisor.
S_FALSE	User is not a supervisor.
Error values	Invalid parameter or unrecognized username.

Remarks

For a user other than the currently logged-on user, this function cannot authenticate that the specified user is really a supervisor. A user interface to obtain the user's password is required for that.

RatingObtainCancel

Allows a browser to cancel its request to obtain the rating for a URL.

HRESULT RatingObtainCancel(
  HANDLE *hRatingObtainQuery* **);**

Parameters

hRatingObtainQuery
  Handle obtained by a previous call to the RatingObtainQuery function.

See Also

RatingObtainQuery

RatingObtainQuery

Encapsulates the process of checking ratings that appear on a local list, third party PICS server, or future third party add-on rating services.

```
HRESULT RatingObtainQuery(
    LPCTSTR pszTargetUrl,
    DWORD dwUserData,
    void (
    *fCallback
    )(
    DWORD dwUserData,
    HRESULT hres,
    LPCTSTR pszRating,
    LPVOID lpvRatingDetails ),

    HANDLE *phRatingObtainQuery
);
```

Parameters

pszTargetUrl
URL whose ratings are to be queried.

dwUserData
Data to be passed to the callback function specified by *fCallback*.

phRatingObtainQuery
Address of a handle that can be used to cancel this operation. This parameter can be NULL if the handle is not needed.

fCallback
Application-defined function that is called upon completion. This function is called in the context of a different thread than the one that called **RatingObtainQuery**.

The callback function has the following parameters:

hres	Result of the query. Can be one of these values:	
	S_OK	Rating found and access allowed.
	S_FALSE	Rating found and access denied.
	Any error values	Rating or service unavailable; look for ratings from other sources if possible.
pszRating	The rating information for the document if it is available. (>Note that the application can choose to ignore this parameter since RatingCheckUserAccess has already been called to generate the *hres* parameter)	
lpvRatingDetails	Token that can be used in the RatingAccessDeniedDialog dialog box. This must be freed with a call to RatingFreeDetails.	

Remarks

A browser obtains the URL from the user and calls this function. Asynchronously, this function searches for a rating for the specified document. When the search ends, the function calls the application's callback function, on a different thread than the one which called **RatingObtainQuery**. While waiting for the callback to be called, browsers may choose to download the content, but prevent the user from

accessing it until the query is resolved.

If **RatingObtainQuery** returns an error, the application should assume the site is unrated by local and third party, and look for the HTML/HTTP level ratings itself.

RatingSetupUI

Displays a modal dialog box that allows a supervising parent to set levels of restrictions. The RatingCheckUserAccess function uses these settings to compare against rating labels.

HRESULT RatingSetupUI(
HWND *hwndParent*,
LPCSTR *pszUsername* **);**

Parameters

hwndParent
Handle of the window handle for the modal dialog box.

pszUsername
Address of a string that contains the name of the user whose settings should be configured. If this parameter is NULL, settings for all users are set. This parameter must be NULL in Internet Explorer version 3.0.

Remarks

This function ensures that the current user is a supervisor before allowing access to the configuration user-interface.

Introduction

Internet Explorer provides new capabilities for application writers to add code to their HTML pages. With that greater power comes greater risk, in the form of malicious code. Internet Explorer introduces some new options and features designed to protect users and provide control over code behavior.

This document describes the potential security hazards of allowing controls to be accessed from scripts and initialized from untrusted data, and discusses hows controls can be marked as safe for these actions so that the user won't receive warning dialogs.

Scripting Security Hazards

Code signing can guarantee that code is trusted to be operated by an end user. However, allowing ActiveX controls to be accessed from scripts raises several new security issues. Even if a control is known to be safe in the hands of a user, it is not necessarily safe when automated by an untrusted script. For example, Microsoft Word is a trusted tool from a (presumably) reputable source, but a malicious script can use its automation model to delete files on the users machine, install macro viruses, and worse.

We therefore require that controls vouch for their ability to be safely scripted. They do this either by supporting a simple interface (IObjectSafety) or through entries in the registry. Such an entry promises that, no matter how malicious a script is, the automation model of the control does not allow any harm to the user, either in the form of data corruption or security leaks. (Of course a control can lie and say its safe for scripting when it isn't. But if control writers really want to be malicious there is no need for scripts to be involved at all—they can do whatever damage it wants directly in the control implementation.)

If a control is not marked as safe for scripting, the user will receive a warning dialog whenever the control is inserted on an untrusted page, asking them whether to the object should be visible from scripts. (This is only at medium security level, at high security, the object is never visible to scripts, and at low, always visible.)

This security mechanism allows controls to be authored which support interesting, potentially unsafe capabilities, while only allowing those capabilities to be used from pages which the user trusts.

Initialization Security Hazards

Another potential security hazard is initializing a control's state using untrusted data. When a control is created, it can be given data from an arbitrary IPersist* interface (from both local and remote URLs) for initializing its state. This again is a potential security hazard because the data could be coming from an untrusted source and the control might do harmful things when given bad or private data.

We therefore define another category of controls—controls which are safe for initializing from persistent data. These are controls which are guaranteed to do nothing bad no matter what data they are initialized with. The user receives a warning, as described above, if a untrusted page attempts to initialize a control that is not safe for initialzing.

Marking a Control as Safe

There are two ways to make a COM object as safe for scripting and initializing—either through a component category entry in the registry, or by supporting the IObjectSafety interface.

Component Category Registry Entry

A control can be marked as safe for scripting or initializing by adding it to the CATID_SafeForScripting and CATID_SafeForInitializing component categories in the registry, respectively. This should be done using the component category manager, which already a standard part of IE3.

Marking Controls As Safe

Use the following code to mark your control as SafeForScripting:

```
hr = CreateComponentCategory(CATID_SafeForScripting, L"Controls that are
safely scriptable");
hr = RegisterCLSIDInCategory(CLSID_SafeItem, CATID_SafeForScripting);
```

or this code to mark your control as SafeForInitializing:

```
hr = CreateComponentCategory(CATID_SafeForInitializing, L"Controls safely
initializable from persistent data");
hr = RegisterCLSIDInCategory(CLSID_SafeItem, CATID_SafeForInitializing);
```

Both of these snippets use the helper functions below, and assume that CLSID_SafeItem contains the CLSID of the control, while CATID_SafeForInitializing and CATID_SafeForScripting are defined in objsafe.h.

RegisterCLSIDInCategory

This function can be used to create a component category and associated description.

```
#include "comcat.h"
HRESULT CreateComponentCategory(CATID catid, WCHAR* catDescription)
  {

    ICatRegister* pcr = NULL ;
    HRESULT hr = S_OK ;

    hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
      NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister, (void**)&pcr);
  if (FAILED(hr))
    return hr;

    // Make sure the HKCR\Component Categories\{..catid...}
    // key is registered
    CATEGORYINFO catinfo;
    catinfo.catid = catid;
    catinfo.lcid = 0x0409 ; // english

  // Make sure the provided description is not too long.
  // Only copy the first 127 characters if it is
  int len = wcslen(catDescription);
  if (len>127)
    len = 127;
    wcsncpy(catinfo.szDescription, catDescription, len);
  // Make sure the description is null terminated
  catinfo.szDescription[len] = '\0';

    hr = pcr->RegisterCategories(1, &catinfo);
  pcr->Release();

  return hr;
  }
```

RegisterCLSIDInCategory

This function can be used to register a CLSID as belonging to a component category.

```
#include "comcat.h"
HRESULT RegisterCLSIDInCategory(REFCLSID clsid, CATID catid)
  {
// Register your component categories information.
    ICatRegister* pcr = NULL ;
    HRESULT hr = S_OK ;
    hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
      NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister, (void**)&pcr);
    if (SUCCEEDED(hr))
    {
       // Register this category as being "implemented" by
       // the class.
       CATID rgcatid[1] ;
       rgcatid[0] = catid;
       hr = pcr->RegisterClassImplCategories(clsid, 1, rgcatid);
    }

    if (pcr != NULL)
        pcr->Release();

  return hr;
  }
```

UnRegisterCLSIDInCategory

This function can be used to unregister a CLSID as belonging to a component category.

```
#include "comcat.h"
HRESULT UnRegisterCLSIDInCategory(REFCLSID clsid, CATID catid)
  {
    ICatRegister* pcr = NULL ;
    HRESULT hr = S_OK ;
    hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
      NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister, (void**)&pcr);
    if (SUCCEEDED(hr))
    {
       // Unregister this category as being "implemented" by
       // the class.
       CATID rgcatid[1] ;
       rgcatid[0] = catid;
       hr = pcr->UnRegisterClassImplCategories(clsid, 1, rgcatid);
    }

    if (pcr != NULL)
        pcr->Release();

  return hr;
  }
```

Calls to these snippets should be inserted in your DLL registration and unregistration routines.

IObjectSafety

The IObjectSafety interface is an interface that allows an object user to ask the object to make itself safe, or to ask the current safety of the object, with regards to various capabilities on the object. The current defined capabilites are INTERFACESAFE_FOR_UNTRUSTED_CALLER and INTERFACESAFE_FOR_UNTRUSTED_DATA, corresponding the "Safe for Scripting" and "Safe for Initializing", but the design allows us to later define other capabilities.

The interface is as follows.

```
// Option bit definitions for IObjectSafety:
#define INTERFACESAFE_FOR_UNTRUSTED_CALLER     0x00000001  // Caller of
interface may be untrusted
#define INTERFACESAFE_FOR_UNTRUSTED_DATA 0x00000002  // Data passed into
interface may be untrusted

// {CB5BDC81-93C1-11cf-8F20-00805F2CD064}
DEFINE_GUID(IID_IObjectSafety, 0xcb5bdc81, 0x93c1, 0x11cf, 0x8f, 0x20, 0x0,
0x80, 0x5f, 0x2c, 0xd0, 0x64);

interface IObjectSafety : public IUnknown
    {
    public:
        virtual HRESULT __stdcall GetInterfaceSafetyOptions(
            /* [in] */ REFIID riid,
            /* [out] */ DWORD __RPC_FAR *pdwSupportedOptions,
            /* [out] */ DWORD __RPC_FAR *pdwEnabledOptions) = 0;

        virtual HRESULT __stdcall SetInterfaceSafetyOptions(
            /* [in] */ REFIID riid,
            /* [in] */ DWORD dwOptionSetMask,
            /* [in] */ DWORD dwEnabledOptions) = 0;

    };
```

SetInterfaceSafetyOptions

The **SetInterfaceSafetyOptions** function makes an object safe for initialization or scripting.

HRESULT SetInterfaceSafetyOptions(
*/** **[in]** ***/ REFIID** *riid***,**
*/** **[in]** ***/ DWORD** *dwOptionSetMask***,**
*/** **[in]** ***/ DWORD** *dwEnabledOptions*
);

Parameters

riid
Specifies the interface ID for the object to be made safe.

dwOptionSetMask
Specifies the options to be changed.

dwEnabledOptions
Specifies the settings for the options that are to be changed. This argument can be one of the following values:

Value	Meaning
INTERFACESAFE_FOR_UNTRUSTED_CALLER	Specifies that the interface identified by *riid* should be made safe for scripting.
INTERFACESAFE_FOR_UNTRUSTED_DATA	Specifies that the interface identified by *riid* should be made safe for untrusted data during initialization.

Return Values

If the function is successful, it returns S_OK. If the argument *riid* specifies an interface that is unknown to the object, this function returns E_NOINTERFACE. If the *dwOptionSetMask* specifies an option that is not supported by the object, this function returns E_FAIL.

Remarks

Typically, a control client would call *SetInterfaceSafetyOptions* before attempting to script or initialize the object, to attempt to require that the object is safe for the desired action. If SetInterfaceSafetyOptions returns failure, it client will display UI to the user to confirm whether the action should be allowed.

SetInterfaceOptions takes an interface (either IDispatch for scripting or IPersist* for initialization), a bitmask of options to change (dwOptionSetMask) and the new settings for those options (dwEnabledOptions). The currently supported capabilities are, as described above, INTERFACESAFE_FOR_UNTRUSTED_CALLER and INTERFACESAFE_FOR_UNTRUSTED_DATA.

GetInterfaceSafetyOptions

The **GetInterfaceSafetyOptions** function retrieves the safety options supported by an object as well as the safety options that are currently set for that object.

HRESULT GetInterfaceSafetyOptions(
*/** **[in]** ***/ REFIID** *riid*,
*/** **[out]** ***/ DWORD __RPC_FAR** **pdwSupportedOptions*,
*/** **[out]** ***/ DWORD __RPC_FAR** **pdwEnabledOptions*
);

Parameters

riid
Specifies the interface ID for a given object.

pdwSupportedOptions
Specifies the options supported for the interface identified by *riid*.

pdwEnabledOptions
Specifies the options currently enabled for the interace identified by *riid*.

Return Values

TBD.

Remarks

GetInterfaceSafetyOptions returns a set of bits in pdwSupportedOptions for each capability that the control knows about, and a set of bits in pdwEnabledOptions for each capability for which the control is currently safe.

For example, my control might say that it knows about INTERFACESAFE_FOR_UNTRUSTED_DATA and INTERFACESAFE_FOR_UNTRUSTED_CALLER, and is currently safe for just INTERFACESAFE_FOR_UNTRUSTED_DATA.

Microsoft Internet Explorer Scripting Object Model

The Microsoft® Internet Explorer scripting object model is a structure for embedding VBScript and JavaScript in HTML documents. The Internet Explorer model is compatible with Netscape Navigator™ with the object model used in the JavaScript™ language. The Microsoft Internet Explorer object model is accessible from any scripting language that is compatible with the ActiveX scripting framework, such Microsoft Visual Basic® Scripting Edition (VBScript). This document provides an overview of the object model, sample code (in VBScript and in JavaScript), and reference information. This document includes descriptions of the methods, properties, and events used with scripting engines in Internet Explorer.

Note All properties and methods that modify the HTML contents must be called during HTML parse time. The code must reside in a script block that runs inline during the loading of the HTML document. This is called *immediate execution* in the ActiveX Scripting Model.

Attaching and Invoking Scripts

There are three ways to attach and invoke scripts in HTML:

- use the <SCRIPT> tag
- use those attributes of HTML elements that support scripts
- use a custom URL type.

Using the SCRIPT Element

Use the SCRIPT element to add scripts to HTML. SCRIPT is an element for embedding script code in a document. Using SCRIPT, the full source code of a script can be included within the document. The SCRIPT element can be used to point to external scripts, using the SRC attribute.

For example, this HTML describes a page with a SCRIPT element which includes code written in VBScript:

```
<SCRIPT language="VBScript">
  Document.write("Hello, Webmaster.")
</SCRIPT>
```

This displays as:

The example in JavaScript would read:

```
<SCRIPT language="JavaScript">
    //... Additional JavaScript statements ...
</SCRIPT>
```

Evaluation of SCRIPT

The SCRIPT element is evaluated when the document is loaded. All code is executed at load time in the order in which it appears in the document. Therefore, any reference to an object must appear in the text *after* the script element in which the object is defined.

The document object's write method can insert both text and objects—such as buttons and ActiveX controls. These objects can be referenced only in a script block following the script block that defined them. You will be able to refer to and to copy references to objects that are the result of a code download. You can invoke any method on an object&151;but only when the object has been downloaded.

Using Scripts as Attributes of HTML Elements

Another way to insert scripts is to use the attributes of HTML elements that support scripts. When these attributes match with events on the elements, the script is executed when the event is occurs. This can be done with HTML elements, such as forms, buttons, or links; however, this method does not work for items inserted using the OBJECT tag.

The following example uses this syntax in Button1 to handle the onClick event. To demonstrate the ability to combine multiple scripting languages on the same page, the scriplet for Button1 is implemented in VBScript and that for Button2 in JavaScript.

```
<form name="Form1">
  <input type="button" name="Button1" value="VB"
  onClick="pressed" language="VBScript">
  <input type="button" name="Button2" value="Java"   onClick="pressed2()"
language="JavaScript">
</form>

<script language="VBSCRIPT">
  sub pressed
    document.Form1.Button1.value="VBScript"
                alert "Pressed the VBScript button"
  end sub
</script>
<script language="JavaScript">
  function pressed2()
  {
    document.Form1.Button2.value="JavaScript"
    alert("Pressed the Java button.")
  }
</script>
```

This displays as:

Note the use of the language attribute on the input tag to indicate the scriplet's language. If no language is specified, the scriplet defaults to the language of the most recently encountered script block. If no script block has been encountered, the language defaults to JavaScript.

The elements FORM, INPUT, BODY, and A support this syntax, but with differing events. See the individual tags referenced later in this document.

An Alternative Using SCRIPT

The SCRIPT element can also be used with the FOR="object" EVENT="eventname" syntax. This method can be used for any named elements, and for any elements inserted using the OBJECT tag. The following example is similar to the previous script example, but it uses a different syntax:

```
<form name="Form1">
  <input type="button" name="Button1" value="Press">
  <script for="Button1" event="onClick" language="VBScript">
    alert "Button has been presed"
    document.Form1.Button1.value="Button has been pressed."
  </script>
</form>
```

This displays as:

Using Scripts in URLs

Scripts can be invoked using the A element combined with a custom URL type. This allows a script to be executed when the user clicks on a hyperlink. This URL type is valid in any context, but is most useful when used with the A element. For example:

```
<A HREF="javascript:alert('hi there')">Click me to see a message.<A>
```

displays an alert message box that contains the text 'hi there'.

Syntax

script-engine:script-code

Executes the script code using the script engine when the URL is resolved. For example, to execute a script when the user clicks on a hyperlink, use:

```
<title> JavaScript example </title>
<A HREF=" javascript:alert(document.title)">Click here to see the title of
the current document..</A>
```

Notice that the script is executed in the context of the current page, which means that document.title evaluates to the document containing the script.

Argument	Type	Description
script-engine	String	A string that names a scripting engine (*must* be JavaScript for Beta 1).
script-code	String	A string that evaluates to a script in the syntax supported by the scripting engine. This script is executed by the scripting engine when the URL is evaluated.

Note This syntax is only supported for JavaScript in the current build of Internet Explorer; in particular, vbscript: will not work in the current build. All scripting engines will be supported in future builds. Also, the JavaScript: syntax is currently supported only from scripts, not when typed into the address bar by users.

Object Hierarchy and Scoping

There are eleven objects to consider in the HTML object model:

- Window
- **Frame**
- History
- Navigator
- Location
- **Script**
- Document
- Link
- Anchor
- Form
- Element

These objects are organized in the following hierarchy (the dotted line following an object indicates that multiple objects may exist):

{ewc msdncd, EWGraphic, grpsweeper 6 /a "sweeper.bmp"}

Each of these objects has its own rules for scoping and containment.

The Window Object

The top level object is a window. Every window contains:

- **Frame** - Array of contained frame windows. Each frame is a window that has its own properties, including a document.
- History - History object for the current window. This object is used to access the history list from the browser.
- Navigator - Navigator object for the current window. The navigator object contains information about the browser application.
- Location - Location object for the current window. Provides information about the location of the window's URL.
- **Script** - Any scripting function defined using the SCRIPT element in the window scope.
- Document - document in the current window.

The window object properties can be referenced directly by scripts while in the window scope. So, for example, script authors do not need to type:

```
window.name
```

to reference the window name; instead, it is sufficient just to type:

```
name
```

Note also that it is possible to call scripts from one window object to another. So, to execute the script myscript in the topmost window, use:

```
top.myscript()
```

The Document Object

The next level down is a document. This object contains:

- Link - an array of hyperlinks found on the given document.
- Anchor - an array of forms found on the given document
- Form - an array of anchors found on the given document

Because scripts live with the window object, not the document object, the script author must type **document.***property* to access document properties. So, to get the title of the document, the author can type:

```
<script language="VBScript">
'...
string1 = document.title     -put the document title into string1
'...
</script>
```

To access the forms in a document, the author can either refer by name or through the form array. So, for the following form:

```
<form name="Form1">
     <input type="button" name="Button1" value="Press ME"
onClick="pressed">
</form>
```

The author can access the object named button1 either by name:

```
<script language="VBScript">
sub pressed
   document.Form1.Button1.value="I've been pressed"  '   access the form by
name
end sub
</script>
```

or by index:

```
<script language="VBScript">
sub pressed
     document.forms(0).Button1.value="I've been pressed"   '   access the
form by index
end sub
</script>
```

The only unusual part of document naming is contained elements that are not form types. Scripts can refer to these elements directly, without using **document**. So, for example, if the authors create an object called myObject, they can reference it directly in script as follows:

```
<object name="myObject" ... >
</object>
```

```
<script language="VBScript">
sub foo
     myObject.color = "green"     - access the form by index
end sub
</script>
```

The Form Object

The final level of scoping is a form. The form object contains:

- Element - the array of objects and intrinsic controls contained in the form.

Scripts can live either in a form object or in a window object. If a script lives outside the form, it needs to access the elements in the form, either by name or through the form array (see the example in "The Document Object"). If, however, the script lives inside the form, it can access the elements in the form directly.

```
<form name="Form1">
     <input type="button" name="Button1" value="Press me">
     <script for="Button1" event="onClick" language="VBScript">
          alert "I've been pressed"
          document.Form1.Button1.value="OUCH"     - as usual, we can use
the fully qualified name
          Button1.value="OUCH"     - because we're in the form Button1 is
scoped as well
     </script>
</form>

<script language="VBScript">
sub foo
     document.Form1.Button1.value="OUCH"     - outside the form, we can
only use the fully qualified name
end sub
</script>
```

window Object

The top level object in the scripting object model is a window. Every window contains:

- **Frame** - Array of frame windows contained by a parent window. Each frame is a window that has its own properties, including a document.
- History - the history object for the current window. This object is used to access the history list from the browser.
- Navigator - the navigator object for the current window. The navigator object contains information about the browser application.
- location - the location object for the current window. Provides information about the location of the window's URL.
- **Script** - any scripting function defined using the SCRIPT element in the window scope.
- Document - the document in the current window.

{ewc msdncd, EWGraphic, grpsweeper 7 /a "sweeper.bmp"}

The window object represents the Internet Explorer window and its methods and properties. Methods and properties of the window object can be called by scripts directly. This means that if you wanted to get the name of the current page, you would use the following script (Notice that the property name does not need a prefix):

```
<script language="VBScript">
     '...
     string1=name          - get the name of the current window
     alert string1          - display that name as an alert
     '...
</script>
```

However, you can access the properties of other window objects without explicitly mentioning the window. For example, to get the name of the current window's parent, you would use:

```
<script language="VBScript">
     '...
     string1=parent.name       - get the name of the parent window
     '...
</script>
```

window events can be hooked to scripts using extensions to the BODY tag. To add scripts to a window event, in the BODY tag at the top of the page, add a script for either the onLoad or onUnload events. In the following example, the *Foo* function is called when the page is loaded:

```
<HTML>
...
<BODY Language="VBS" onLoad="Foo">
...
<SCRIPT language="VBScript">
...
Sub Foo
  MsgBox "This is sub foo"
End Sub
```

```
...
</SCRIPT>
....
</BODY></HTML>
```

To access a window by name, the window must be given a name. This can happen in three ways: by using the *window.***open** method, by creating the window with a name using the FRAMESET element, or by creating the window with a URL using the TARGET attribute.

The following examples all create a window named *foo* with contents *a.htm*.

```
<SCRIPT Language="VBScript">
window.open ( "a.htm", "foo");
</SCRIPT>

<FRAMESET cols = "200, *" frameborder=0>
  <FRAME name = "foo" src="a.htm">
  <FRAME name = "bar" src="b.htm">
</FRAMESET>

<A HREF="a.htm" TARGET = "foo">Click here to see a.htm in window foo.</A>
```

The current implementation of Internet Explorer does not support *window.***open**.

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

Properties

window properties can be referenced directly in the scripting language. Consequently, all window properties are reserved words and cannot be used as variable names in procedures. The following window properties are used:

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

name Property

Returns the name of the current window.

[*window.*]**name**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns the string containing the current window name. Note that the current implementation always returns "Microsoft Internet Explorer."

Remarks

To set the value of String1 to be the name of the current window, use:

```
String1=name.
```

This property is read-only.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

parent, self, top, location, defaultStatus, status, frames, history, navigator, document

parent Property

Returns the window object of the window's parent. This property is read-only. The parent of the window is the containing frame. If the current window has no containing frame windows, then the parent evaluates to the current window.

[*window.*]**parent**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns the window object that evaluates to the parent window.

Remarks

To set the value of String1 to be the name of the parent of the current window, use:

```
String1=parent.Name.
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, self, top, location, defaultStatus, status, frames, history, navigator, document

self Property

Returns the window object of the current window. This property is read-only.

[*window.*]**self**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object that evaluates to the current window.

Remarks

To set the value of String1 to be the name of the current window, use:

```
String1=self.name
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, top, location, defaultStatus, status, frames, history, navigator, document

top Property

Returns the window object of the topmost window. This property is read-only. The topmost window is the containing window of all frames in the current browser instance.

[*window.*]**top**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object that evaluates to the topmost window.

Remarks

To set the value of String1 to be the name of the topmost window, use:

```
String1=top.name.
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, location, defaultStatus, status, frames, history, navigator, document

location Property

Returns the location object for the current window. This property is read-only. For more details, see "Location Object."

[*window.*]**location**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object that evaluates to the location object of window.

Remarks

To set the value of String1 to be the name of the URL of the current window, use:

```
String1=location.HRef.
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, defaultStatus, status, frames, history, navigator, document

defaultStatus Property

Gets or sets the default status text in the lower left portion of the status bar.

[*window.*]**defaultStatus**[**=***string*]

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

String
Optional. Sets the default status text to the value of String.

Return Values

Returns the default status text.

Remarks

To set the default status to "Hello," use:

```
defaultStatus="Hello"
```

Note that this property does not currently set the default status message, so it is the same as calling **status**.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, status, frames, history, navigator, document

status Property

Gets or sets the status text in the lower left of the status bar.

[*window.*]**status**[**=***string*]

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

String
Optional. Sets the status text to the value of String.

Return Values

Returns the current status text.

Remarks

To set the status to "Hello," use:

```
status="Hello."
```

Currently not implemented.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, frames, history, navigator, document

frames Property

Returns the array of frames for the current window.

[*window.*]**frames**[*integer*]

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object expression that evaluates to the array of frames.

Remarks

To set String1 to the URL of frame[0], use:

```
String1=parent.frames[0].location.href.
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, history, navigator, document

history Property

Returns the history object of the current window. For more details on methods, properties, and events, see "history Object."

[*window.*]**history**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object expression that evaluates to a history object.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, navigator, document

navigator Property

Returns the navigator object of the current window. For more details on methods, properties, and events, see "navigator Object."

[*window.*]**navigator**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object expression that evaluates to a navigator object.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, document

document Property

Returns the document object of the current window. For more details on methods, properties, and events, see "document Object."

[*window.*]**document**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object expression that evaluates to a document object.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator

Methods

This section describes the methods for the window object.

alert Method

Displays an alert message box.

[*window.*]**alert** *string*

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

String
A string containing the text to display in the message box.

Remarks

The following example would display an alert that contained the string "Hello World":

```
Alert "Hello World"
```

Applies To

Window

Methods

confirm, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

confirm Method

Displays a message box that allows the user to select **OK** or **Cancel** and returns either TRUE or FALSE.

[*bool* **=**][*window.*]**confirm** *string*

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

String
A string containing the text to display in the message box.

Return Values

Returns the user response: TRUE if the user pressed OK; FALSE if not.

Remarks

The following example would display a message box that contained the string "Do you want to continue?":

```
x=Confirm "Do you want to continue?"
```

Applies To

Window

Methods

alert, prompt, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

prompt Method

Prompts the user for input.

[*string* **=**][*window.*]**prompt** [*prompt*] [**,** *default*]

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

String
Optional. A string containing the text to display in the message box.

default
Optional. A string containing the default text to display in the input field.

Return Values

Returns the user input.

Not implemented in current build.

Applies To

Window

Methods

alert, confirm, open, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

open Method

Creates a new window.

[*newwindow* **=**][*window.*]**open** *url***,** *target***,** **["[toolbar=***bool***]** **[, location=***bool***][, directories=***bool***][, status=***bool***][, menubar=***bool***][, scrollbars=***bool***][, resizeable=***bool***][, width=***pixels***][, height=***pixels***]"]**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

url
A string containing a correctly parsed URL. The URL is parsed identically to a link—both relative and absolute paths are supported.

target
A string containing the name of the target window. If a window with this name already exists, the existing window is reused with the new URL. If the window does not exist, a new window is created with that name. Note that this works identically to the TARGET attribute of an HREF in HTML.

bool
The remaining window properties are passed as a comma-separated list. Most of these can be set to Boolean values, either *yes* or *no* [*1* or *0*]. These properties are toolbar, location, directories, status, menubar, scrollbars, and resizeable.

pixels
Two other properties in this list, width and height, have values in pixels.

Return Values

Returns an object expression that evaluates to the created window object.

Remarks

The following example would create a new window:

```
 open "http://www.microsoft.com", "myWindow", "toolbar=no, location=no,
directories=no"
```

Note: This feature is not currently implemented in Internet Explorer.

Applies To

Window

Methods

alert, confirm, prompt, close, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

close Method

Closes the window.

[*window.*]**close**

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Return Values

Returns an object expression that evaluates to the indexed frame.

Remarks

Note: This feature is not currently implemented in Internet Explorer.

Applies To

Window

Methods

alert, confirm, prompt, open, setTimeout, clearTimeout, navigate

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

setTimeout Method

Sets a timer to call a function after a specified number of milliseconds.

ID = [*window.*]**setTimeout** *expression*, *msec*

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Expression
An object expression that evaluates to a function or object property. This function is called when the Timeout is set.

MSec
The number of milliseconds that passes before the expression is evaluated.

Return Values

Returns the ID of the timer object. This can be used to cancel the timer using the **clearTimeout** method.

Remarks

To call Button1.Click after 100 milliseconds, use:

```
MyID = setTimeout ("Button1.Click", 100).
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, clearTimeout, navigate

Events

onLoad, onUnload

Properties

parent, self, top, location, defaultStatus, status, frames, name, history, navigator, document

clearTimeout Method

Clears the timer having a particular ID.

[*window.*]**clearTimout** *ID*

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

ID
The ID of the timer to be cleared. If there is no timer with this ID, the function does nothing.

Remarks

To clear the timer with ID=MyID, use.

```
clearTimeout MyID
```

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, navigate

Events

onLoad, onUnload

Properties

parent, self, top, location, defaultStatus, status, frames, name, history, navigator, document

navigate Method

Navigates the window to a new URL.

[*window.*]**navigate** *url*

Parts

window
Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

url
A string containing a valid URL. The URL can be either relative or absolute.

Applies To

Window

Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout

Events

onLoad, onUnload

Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

onLoad Event

Fired when the contents of the window are loaded.

onLoad=*function-name*

Values

function-name
An object expression that evaluates to a scripting function.

Remarks

To call the VBS function Foo when the page is loaded, use:

```
<BODY Language="VBS" onLoad="Foo">
```

Applies To

Window

onUnload Event

Fired when the contents of the window are unloaded.

onUnload=*function-name*

Values

function-name
An object expression which evaluates to a scripting function.

Remarks

To call the VBS function Foo when the page is unloaded, use:

```
<BODY Language="VBS" onUnload="Foo">[<window.>]Navigate url
```

Applies To

Window

document Object

An object that resides below the window in the scripting object model. A document may contain:

- Link - an array of hyperlinks found on the given document
- Anchor - an array of anchors found on the given document
- Form - an array of forms found on the given document

{ewc msdncd, EWGraphic, grpsweeper 8 /a "sweeper.bmp"}

The document object reflects the HTML document currently in the browser and objects on the page—that is, links, forms, buttons, and ActiveX Objects. Methods and properties of the document object must be called in a script by placing *document* first in the statement. This means that if you wanted to set the background color on the page, the script would look like:

```
<script language="VBScript">
     document.bgColor = "Blue"
</script>
```

The document object currently has no events.

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

linkColor Property

Gets or sets the current color of the links in a document.

document.**linkColor** [**=***rgb-value*|*string*]

Parts

document
An object expression that evaluates to a document object.

rgb-value
Optional. The new color of links in the document.

string
Optional. A string value specifying the color.

Return Values

Returns the rgb value of the current link color.

Remarks

Note that this property can only be set at parse time, not after the page is painted. So the code:

```
<SCRIPT LANGUAGE="JavaScript">
document.vLinkColor = "green";
document.linkColor = "red";
document.alinkColor = "aqua";
</SCRIPT>
```

sets the link color, while the code:

```
<FORM>
"document.linkColor='#000000'">
<INPUT TYPE="button" VALUE="Set Visited Link Color to White" onClick =
"document.vLinkColor='#FFFFFF'">
</FORM>
```

will have no effect when the button is clicked. The performance hit of changing the link color after parse time is simply too great.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

aLinkColor Property

Gets or sets the current color of the *active* links in a document. A link is *active* when the mouse pointer is held down over the link but not released. Note that Internet Explorer does not have this feature, so **aLinkColor** has no effect; however, it is supported in the object model for compatibility reasons. As with **linkColor**, this property can only be set at parse time. For details, see the examples in **linkColor**.

*document.***aLinkColor** [**=***rgb-value*|*string*]

Parts

document
An object expression that evaluates to a document object.

rgb-value
Optional. The new color of links in the document.

string
Optional. A string value specifying the color.

Return Values

Returns the rgb value of the current link color.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

vLinkColor Property

Gets or sets the current color of the visited links in a document. As with **linkColor**, this property can only be set at parse time. See the examples in **linkColor** for details.

document.**vLinkColor** [**=***rgb-value*|*string*]

Parts

document
An object expression that evaluates to a document object.

rgb-value
Optional. The new color of links in the document.

string
Optional. A string value specifying the color.

Return Values

Returns the rgb value of the current link color.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

bgColor Property

Gets or sets the current color of the background in a document.

document.**bgColor** [**=***rgb-value*|*string*]

Parts

document
An object expression that evaluates to a document object.

rgb-value
Optional. The new color of the background in the document.

string
Optional. A string value specifying the color.

Return Values

Returns the rgb value of the current background color.

Remarks

To set the background color to white, use:

```
document.bgColor="000000"
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

fgColor Property

Gets or sets the foreground color.

document.**fgColor**[**=***rgb-value*]

Parts

document
An object expression that evaluates to a document object.

rgb-value
Optional. The new color of the foreground in the document.

Remarks

To set the foreground color to white, use:

```
document.fgColor="000000".
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

anchors Property

Returns the array of anchors in a document.

document.**anchors**[*integer*]

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns an object expression that evaluates to the array of anchors.

Remarks

To access the first anchor in the document, use:

```
document.anchors[0]
```

To get the length of the anchors array, use:

```
document.anchors.length
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, links, forms, location, lastModified, title, cookie, referrer

links Property

Returns the array of links for the current document.

document.**links** [*integer*]

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns an object expression that evaluates to the array of links.

Remarks

To access the first link in the document, use:

```
document.Links[0]
```

To get the length of the links array, use:

```
document.links.length
```

Note that the locations in the links collection are read-only in the current build. In future builds you will be able to reset the targets of links.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, forms, location, lastModified, title, cookie, referrer

forms Property

Returns the array of forms in a document.

document.**forms** [*integer*]

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns an object expression that evaluates to the array of forms.

Remarks

To access the first form in the document, use:

```
document.Forms[0]
```

To get the length of the forms array, use:

```
document.forms.length
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, location, lastModified, title, cookie, referrer

location Property

Returns a read-only representation of the location object.

document.**location**

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns an object expression that evaluates to the location object of the document.

Remarks

To set String1 to the document's URL, use:

```
String1 = document.location.Href
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, lastModified, title, cookie, referrer

lastModified Property

Returns the last modified date of the current page.

document.**lastModified**

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns a string containing the date.

Remarks

To set Date1 to the document's URL, use:

```
Date1 = document.lastModified
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, title, cookie, referrer

title Property

Returns a read-only representation of the document's title.

document.**title**

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns an object expression that evaluates to the location object of the document.

Remarks

To set String1 to the document's title, use:

```
String1 = document.title
```

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, cookie, referrer

cookie Property

Gets or sets the cookie for the current document.

document.**cookie** [**=***newcookie*]

Parts

document
An object expression that evaluates to a document object.

newcookie
Optional. The new value for the cookie. Because the cookie file is just a text file, this value is a string.

Return Values

Returns a string containing the current cookie.

Remarks

The cookie is a string expression stored for the current page. Note that setting the cookie overwrites any current cookie information. Also note that you can use string expressions to locate particular information in the cookie string.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, referrer

referrer Property

Gets the URL of the referring document.

*document.***referrer**

Parts

document
An object expression that evaluates to a document object.

Return Values

Returns a string containing the URL of the referring document.

Currently returns the URL of the referring document when there is a referrer, and NULL when there is no referrer.

Remarks

The referring document is the document that contained the link the user clicked on to get to the current document. For example, if the user is on www.microsoft.com and clicks on a link to navigate to www.msn.com, the referrer property of the document for www.msn.com is www.microsoft.com. Note that by definition the referrer varies depending on how the user linked to the current document. If the user navigated to the document without clicking on a link from another page, referrer should return NULL.

Applies To

Document

Methods

write, writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie

write Method

Places the given string into the current document. Unless otherwise specified, the string is appended to the current document at the current position.

document.**write***string*

Parts

document
An object expression that evaluates to a document object.

string
The string to write to the current document. Note that the string is added into the HTML directly, so it must be formatted as HTML.

Remarks

The following examples demonstrate the use of the **write** method:

```
<HTML><BODY>
<SCRIPT LANGUAGE='VBS'>
document.Write ("Hello world.")
</SCRIPT>
This is a document.
</BODY></HTML>
```

results in:

```
Hello world. This is a document.
```

Whereas:

```
<HTML><BODY>
This is a document.
<SCRIPT LANGUAGE='VBS'>
document.Write ("Hello world.")
</SCRIPT>
</BODY></HTML>
```

results in:

```
This is a document. Hello world.
```

Applies To

Document

Methods

writeLn, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

writeLn Method

Places the given string into the current document with a new-line character appended to the end.

*document.***writeLn** *string*

Parts

document
An object expression that evaluates to a document object.

string
The string to write to the current document. Note that the string is added into the HTML directly, so it must be formatted as HTML.

Remarks

This method is the same as the *document.***write** method with the addition of a newline character at the end. Note that a newline is ignored by HTML unless it is bracketed by <PRE> tags, so in many cases *document.***write** and *document.***writeLn** behave exactly the same.

The following examples demonstrate the use of the **writeLn** method:

```
<SCRIPT LANGUAGE='VBS'>
document.writeLn ("Hello world.")
document.write ("Hello world.")
</SCRIPT>
```

results in:

```
Hello world.  Hello world.
```

Whereas:

```
<PRE>
<SCRIPT LANGUAGE='VBS'>
document.writeLn ("Hello world.")
document.write ("Hello world.")
</SCRIPT>
</PRE>
```

results in:

```
Hello world.
Hello world.
```

Applies To

Document

Methods

write, open, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

open Method

Opens the document stream for output.

*document.***open** [*mimeType*]

Parts

document
An object expression that evaluates to a document object.

mimeType
Optional. A string containing a valid mime type. Note that this can include types supported by Internet Explorer (text/html, text/plain), but can also include other types (application/x-director for MacroMedia director). In the case of other types, Internet Explorer creates a file containing the data between *document.***open** and *document.***close** calls, and hands the created file to the correct application.

Remarks

This method is implemented in the current build; however, the mime-type is currently ignored (html is always assumed).

Generally *document.***open** is followed by a sequence of *document.***write** or *document.***writeLn** statements, followed by *document.***close**. If the referenced document exists already, any information contained in the document is cleared. To write "Hello World" to the document, use:

```
document.open
document.writeLn "Hello World"
document.close
```

Note that this is identical to:

```
document.writeLn "Hello World"
```

with two exceptions.

- In the first example, "Hello World" is written to the screen after *document.***close**; in the second, it is written immediately.
- In the first example, *document.***open** clears the document if there is data; in the second, "Hello world" is appended to the end.

Applies To

Document

Methods

write, writeLn, close, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

close Method

Updates the screen to display all of the strings written after the last open method call.

*document.***close**

Parts

document
An object expression that evaluates to a document object.

Applies To

Document

Methods

write, writeLn, open, clear

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

clear Method

Closes the document output stream and writes the data to the screen. See the open method description for more information and examples.

*document.***clear**

Parts

document
An object expression that evaluates to a document object.

Remarks

Not implemented in current build.

Applies To

Document

Methods

write, writeLn, open, close

Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

form Object

An object that resides below the document in the scripting object model. A form may contain:

- Element - The array of objects and intrinsic controls contained in the form.

{ewc msdncd, EWGraphic, grpsweeper 9 /a "sweeper.bmp"}

The form object represents a form in the HTML document. Forms are kept in the document object both as an array and by name. Script forms are accessible either by index (the documents forms array) or by name (given in the NAME="somename" attribute of the HTML <FORM> tag). Given a document with one form defined, the script can access the form in one of two ways:

```
<script language="VBSCRIPT">

' ...first method, by name ...
sub pressedByName
     document.Form1.Button1.value="I've been pressed"     ' access the form
by name
end sub

' ... second method, by index...
' Note that indexes start at 0, not 1!
sub pressedByIndex
     document.form1.elements(1).value="I've been pressed"     ' access the
form by index
end sub
</script>

<form name="Form1">
     <input type="button" name="Button1" value="Press ME"
onClick="pressedByName" language="VBScript">
     <input type="button" name="Button2" value="Press ME"
onClick="pressedByIndex" language="VBScript">
</form>
```

Methods

submit

Events

onSubmit

Properties

action, encoding, method, target, elements, hidden

action Property

Gets or sets the address to be used to carry out the action of the form.

*form.***action**[**=***string*]

Parts

form
An object expression that evaluates to a form object.

string
Optional. A string containing the new action, generally a URL.

Return Values

Returns a string containing the current form action.

Remarks

If no URL is specified, the base URL of the document is used. Note that this is identical to changing the ACTION attribute of the FORM tag. So the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search">
</FORM>
```

Applies To

Form

Methods

submit

Events

onSubmit

Properties

encoding, method, target, elements, hidden

encoding Property

Gets or sets the encoding for the form.

*form.***encoding**[**=***string*]

Parts

form
An object expression that evaluates to a form object.

string
Optional. A string containing the new encoding. This must be a valid mime type, like "text/html".

Return Values

Returns a string containing the current form encoding.

Remarks

If no mime type is specified, "text/html" is used. Note that this is identical to changing the ENCTYPE attribute of the FORM tag. So the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"
document.form[0].enctype = "text/html"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" ENCTYPE="text/html">
</FORM>
```

Note that in the current build, encoding can be set, but has no effect on the operation of the form.

Applies To

Form

Methods

submit

Events

onSubmit

Properties

action, method, target, elements, hidden

method Property

Indicates how the form data should be sent to the server.

*form.***method**[*string*]

Parts

form
An object expression that evaluates to a form object.

string
Optional. A string containing the new method, either GET or POST.

Return Values

Returns a string containing the current form method.

Remarks

GET means append the arguments to the action URL and open it as if it were an anchor; POST means send the data via an HTTP post transaction. Note that this is identical to the METHOD attribute of the FORM tag, so the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"
document.form[0].method = "GET"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" METHOD=GET>
</FORM>
```

Applies To

Form

Methods

submit

Events

onSubmit

Properties

action, encoding, target, elements, hidden

target Property

Specifies the name of the target window to display the form results in.

*form.***target** [**=***string*]

Parts

form
An object expression that evaluates to a form object.

string
Optional. A string containing the new target name.

Return Values

Returns a string containing the current form target.

Remarks

Note that this is identical to the TARGET attribute of the FORM tag, so the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"
document.form[0].target = "newWindow"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" TARGET="newWindow">
</FORM>
```

Note that in the current build, **target** can be set; however, it has no effect on the operation of the form.

Applies To

Form

Methods

submit

Events

onSubmit

Properties

action, encoding, method, elements, hidden

elements Property

Returns the array of elements contained in the form.

*form.***elements**[**=***string*]

Parts

form
An object expression that evaluates to a form object.

Return Values

Returns an object expression that evaluates to the array of elements in a form.

Remarks

The elements include any intrinsics (specified using the INPUT tag) or any embedded objects (specified using the OBJECT tag) contained in the form. So, the HTML:

```
<FORM ACTION="http:// www.sample.com/bin/search" METHOD=GET>
<INPUT NAME="aButton" TYPE ... >
<INPUT NAME="aCheckBox" TYPE ... >
<OBJECT NAME="anObject" DATA=...></OBJECT>
<INPUT NAME="aRadio" TYPE ... >
</FORM>
```

would generate an elements array where *form.***elements**.length returns 4, and *form.***elements**[2].name returns "anObject".

Applies To

Form

Methods

submit

Events

onSubmit

Properties

action, encoding, method, target, hidden

hidden Property

Not yet implemented

Applies To

Form

Methods

submit

Events

onSubmit

Properties

action, encoding, method, target, elements

submit Method

Submits the form. Note that this is identical to clicking a form input with TYPE=SUBMIT.

*form.***submit**

Parts

form
An object expression that evaluates to a form object.

Applies To

Form

Events

onSubmit

Properties

action, encoding, method, target, elements, hidden

onSubmit Event

Fired when the form is submitted.

*form.***onSubmit =***action*

Values

form
An object expression that evaluates to a form object.

action
A string expression that evaluates to a scripting function call.

Remarks

This event can be used to prevent the form from being submitted, or it can be used simply to run additional code before the form is submitted. To prevent the form from being submitted, you must use "return <function>." So, the script:

```
form.onsubmit = "return IsValid()"
```

calls IsValid, and submits the form if it returns TRUE, or doesn't submit the form if it returns FALSE, while:

```
form.onsubmit = "IsValid()"
```

calls IsValid, but submits the form regardless of return value.

In the current build, forms fire the **onSubmit** event when the submit method is called; but not when the submit button is clicked.

Applies To

Form

Methods

submit

Properties

action, encoding, method, target, elements, hidden

location Object

An object that resides below the document in the scripting object model. The location object represents the current URL:

{ewc msdncd, EWGraphic, grpsweeper 10 /a "sweeper.bmp"}

Setting any portion of the location object causes the browser to navigate to the newly constructed URL. The following script navigates to http://www.microsoft.com:

```
<script language="VBScript">
     [some preceding VBScript code]
     location.href=" http://www.microsoft.com"
</script>
```

Properties

href, protocol, host, hostname, port, pathname, search, hash

href Property

Gets or sets the compete URL for the location.

*location.***href** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the complete URL for the location.

Applies To

Location

Properties

protocol, host, hostname, port, pathname, search, hash

protocol Property

Gets or sets the protocol portion of the URL.

location.**protocol** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the protocol portion of the URL.

Remarks

For http://www.microsoft.com, this would return http:.

Applies To

Location

Properties

href, host, hostname, port, pathname, search, hash

host Property

Gets or sets both the host and port portion of the URL (hostname:port.).

*location.***host** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the host and port portion of the URL.

Remarks

For http://www.microsoft.com, this would be www.microsoft.com:80. For file: protocols, this always returns "".

Applies To

Location

Properties

href, protocol, hostname, port, pathname, search, hash

hostname Property

Gets or sets the host portion of the URL, either a name or an IP address.

*location.***hostname** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the hostname portion of the URL.

Remarks

For http://www.microsoft.com, this would return www.microsoft.com. For file: protocols, this always returns "".

Applies To

Location

Properties

href, protocol, host, port, pathname, search, hash

port Property

Gets or sets the port of the URL.

location.**port** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the port of the URL.

Remarks

For http://www.microsoft.com, this returns 80. For file: protocols, this always returns "".

Applies To

Location

Properties

href, protocol, host, hostname, pathname, search, hash

pathname Property

Gets or sets the pathname in the URL.

location.**pathname** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the pathname portion of the URL.

Note that the current implementation returns "intdev", not "/intdev" as expected.

Remarks

For http://www.microsoft.com/intdev, this returns intdev.

Applies To

Location

Properties

href, protocol, host, hostname, port, search, hash

search Property

Gets or sets the search portion of the URL, if specified.

location.**search** [**=***string*]

Parts

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the search portion of the URL.

Remarks

For http://www.microsoft.com/intdev?user, this returns ?user. For http://www.microsoft.com/intdev, this returns NULL.

Applies To

Location

Properties

href, protocol, host, hostname, port, pathname, hash

hash

Gets or sets the hash portion of the URL, if specified.

*location.***hash** [**=***string*]

Parameters

location
An object expression that evaluates to a location object.

string
Optional. The new string value.

Return Values

Returns a string containing the hash portion of the URL.

Note that current implementation returns "#" always.

Applies To

Location

Properties

href, protocol, host, hostname, port, pathname, search

link Object

An object that resides below the document in the scripting object model. This object specifies an array of links for a given document.

{ewc msdncd, EWGraphic, grpsweeper 11 /a "sweeper.bmp"}

The link object is referenced as a read-only property array. A link object is constructed for every link that appears in the HTML document. A link is defined in scripting as the anchor tag <A> containing the HREF attribute <A HREF="http://www.microsoft.com">. All properties of the link object are read-only and are the same as the location object's properties. It is only accessible through the indexed array. The following lines of script would set linktext to the third link on the page (if it exists):

```
<script language="VBScript">
  [some preceding VBScript code]
  linktext = document.links(2).href
  [some following VBScript code]
</script>
```

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, port, pathname, search, hash, target

href Property

Returns the compete URL for the link.

link.**href**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the complete URL for the link.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

protocol, host, hostname, port, pathname, search, hash, target

protocol Property

Returns the protocol portion of the URL.

*link.***protocol**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the protocol portion of the URL.

Remarks

For http://www.microsoft.com, this would return http:.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, host, hostname, port, pathname, search, hash, target

host Property

Returns both the host and port portion of the URL (hostname:port).

*link.***host**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the host and port portion of the URL.

Remarks

For http://www.microsoft.com, this would return www.microsoft.com:80.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, hostname, port, pathname, search, hash, target

hostname Property

Returns the host portion of the URL, either a name or an IP address.

*link.***hostname**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the hostname portion of the URL.

Remarks

For http://www.microsoft.com, this would return www.microsoft.com.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, port, pathname, search, hash, target

port Property

Returns the port of the URL.

*link.***port**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the port of the URL.

Remarks

For http://www.microsoft.com, this returns 80 (the default for HTTP).

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, pathname, search, hash, target

pathname Property

Returns the pathname in the URL.

*link.***pathname**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the pathname portion of the URL.

Remarks

For http://www.microsoft.com/intdev, this returns /intdev.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, port, search, hash, target

search Property

Returns the search portion of the URL, if specified.

link.**search**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the search portion of the URL.

This returns "user," not "?user," in the current implementation; the leading '?' is omitted.

Remarks

For http://www.microsoft.com/intdev?user, this returns user.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, port, pathname, hash, target

hash Property

Returns the hash portion of the URL, if specified.

*link.***hash**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the hash portion of the URL.

This returns NULL in the current implementation when no hash is specified..

Remarks

This is the section of the URL after # including the #. For http://www.microsoft.com/intdev#user, this returns #user. If no hash is specified, this property returns NULL.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, port, pathname, search, target

target Property

Returns the target of the link, if specified.

link.**target**

Parts

link
An object expression that evaluates to a link object.

Return Values

Returns a string containing the target of the link.

Remarks

This is the same as the value of the TARGET attribute of the LINK tag.

Applies To

Link

Events

mouseMove, onMouseOver, onClick

Properties

href, protocol, host, hostname, port, pathname, search, hash

Events

Link events can be used to set status bar text or other custom actions on mouse movement. The following example is an excerpt from an HTML document that uses a text control to display rich information about the links in an image map. The code decides on the link location.

```
<script language="VBScript" for="Link1" event="mouseMove(shift, button, x,
y)">
    if (InRect(x, y,  5, 30, 120, 85)=true) then
    DescribeLink "A full description of Microsoft's product line"
    [some following VBScript code]
</script>
```

mouseMove Event

Fires an event any time the pointer moves over a link.

link.**mouseMove** *shift*, *button*, *x*, *y*

Values

link
An object expression that evaluates to a link object.

shift
The status of the shift key.

button
Indicates which button is pressed, if any.

x
The horizontal position of the pointer, in pixels.

y
The vertical position of the pointer, in pixels.

Remarks

Shift and button are currently set to zero. x and y contain the actual positional data. To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="mouseMove(shift,
button, x, y)">
```

Applies To

Link

Events

onMouseOver, onClick

Properties

href, protocol, host, hostname, port, pathname, search, hash, target

onMouseOver Event

Fires an event any time the pointer moves over a link.

link.**onMouseOver**

Values

link
An object expression that evaluates to a link object.

Remarks

Not implemented in current builds.

To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="onMouseOver">
```

or attach a script directly in the HTML:

```
<A HREF="http://www.microsoft.com" onMouseOver="alert ('Clicked here')">To
Microsoft</A>
```

Applies To

Link

Events

mouseMove, onClick

Properties

href, protocol, host, hostname, port, pathname, search, hash, target

onClick Event

Fires an event any time you click on a link.

link.**onClick**

Values

link
An object expression that evaluates to a link object.

Remarks

Not implemented in current builds.

To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="onClick">
```

or attach a script directly in the HTML:

```
<A HREF="http://www.microsoft.com" onClick="alert ('Clicked here')">To
Microsoft</A>
```

Applies To

Link

Events

mouseMove, onMouseOver

Properties

href, protocol, host, hostname, port, pathname, search, hash, target

anchor Object

An object that resides below the document in the scripting object model. This object specifies an array of anchors for a given document. Each entry in this array corresponds to an anchor <A> tag that is found in the corresponding document.

{ewc msdncd, EWGraphic, grpsweeper 12 /a "sweeper.bmp"}

The anchor object is referenced as a read-only property array. An anchor object is constructed for every anchor tag <A> found in the HTML document. It is only accessible through the indexed array. The following lines of script would set anchortext to the name of the third anchor on the page (if it exists).

```
<script language="VBScript">
     [some preceding VBScript code]
     anchortext = document.anchors(2).name
     [some following VBScript code]
</script>
```

Properties

name

name Property

Gets or sets the name of the anchor.

*anchor.***name** [**=***string*]

Parts

anchor
An object expression that evaluates to an anchor object.

string
A string containing the new anchor name.

Return Values

Returns a string containing the complete name of the anchor.

element Object

An object that resides below the document in the scripting object model. Elements are intrinsic HTML controls or objects. Controls are placed on a document with the <INPUT> tag while objects are placed on a document with the <OBJECT> tag.

{ewc msdncd, EWGraphic, grpsweeper 13 /a "sweeper.bmp"}

Elements are intrinsic HTML controls (placed on a page through the input tag <INPUT>) or objects that are insertable in HTML via the object tag <OBJECT>. These include ActiveX Controls. They can be referenced either by array or name, but this reference must follow the form identifier. Not all properties, methods, and events apply to all elements. Some properties apply to all elements; some only apply to specific elements. See the list below for details by element type, then see the specific method, event, or property documentation for details.

Element	Properties	Methods	Events
button, reset, submit	form, name, value, enabled	click, focus	onClick, onFocus
check box	form, name, value, checked, defaultChecked, enabled	click, focus	onClick, onFocus
radio	form, name, value, checked, enabled	click, focus	onClick, onFocus
combo	form, name, value, enabled, listCount, list, multiSelect, listIndex	click, focus, removeItem, addItem, clear	onClick, onFocus
password	form, name, value, defaultValue, enabled	focus, blur, select	onFocus, onBlur
text, text area	form, name, value, defaultValue, enabled	focus, blur, select	onFocus, onBlur, onChange, onSelect
select	name, length, options, selectedIndex	focus, blur	onFocus, onBlur, onChange
hidden	name, value		

Methods

click, focus, blur, select, removeItem, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount,

multiSelect, listIndex, length, options, selectedIndex

form Property

Gets the form object containing the element.

*element.***form**

Parts

element
Returns an object expression that evaluates to an intrinsic control.

Return Values

An object expression that evaluates to the form containing the element

Applies To

All elements

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

name Property

Gets or sets the name of the element.

*element.***name** [**=***string*]

Parts

element
An object expression that evaluates to an intrinsic control.

string
Optional. A string containing the new element name.

Return Values

Returns a string containing the name of the element.

Applies To

All elements

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onFocus, onBlur, onChange, onSelect

Properties

form, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

value Property

Gets or sets the value of the element.

*element.***value** [**=***string*]

Parts

element
An object expression that evaluates to an intrinsic control.

string
Optional. A string containing the new element value.

Return Values

Returns a string containing the value of the element.

Applies To

All elements

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

defaultValue Property

Gets or sets the default value of the element.

*element.***defaultValue** [**=***string*]

Parts

element
An object expression that evaluates to an intrinsic control.

string
Optional. A string containing the new default value.

Return Values

Returns a string containing the default value of the element.

Applies To

password, **text**, **text area**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

checked Property

Gets or sets the checked state of the check box or the radio button.

*element.***checked** [**=***bool*]

Parts

element
An object expression that evaluates to an intrinsic control.

string
Optional. Sets the checked state of the check box or the radio button.

Return Values

Returns TRUE if the check box or radio button is checked; FALSE if not.

Applies To

check box, **radio button**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

defaultChecked Property

Gets or sets the default checked property of the check box.

*element.***defaultChecked** [**=***bool*]

Parts

element
An object expression that evaluates to an intrinsic control.

string
Optional. Sets the default state of the check box.

Return Values

Returns TRUE if the check box is checked by default; FALSE if not.

Applies To

check box

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

enabled Property

Gets or sets whether the control is enabled.

*element.***enabled** [**=***bool*]

Parts

element
An object expression that evaluates to an intrinsic control.

bool
Optional. Enables or disables the control.

Return Values

Returns TRUE if the control is enabled; FALSE if not.

Applies To

All elements

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, listCount, multiSelect, listIndex, length, options, selectedIndex

listCount Property

Gets the count of elements in the list.

*element.***listCount**

Parts

element
An object expression that evaluates to an intrinsic control.

Return Values

Returns the number of elements in the combo box.

Applies To

combo

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, multiSelect, listIndex, length, options, selectedIndex

multiSelect Property

Gets or sets whether the combo is multiselect or not.

*element.***multiSelect** [**=***bool*]

Parts

element
An object expression that evaluates to an intrinsic control.

bool
Optional. Use TRUE to set the combo to multiselect; FALSE to set to single-select.

Return Values

Returns TRUE if the combo is multiselect; FALSE if not.

Applies To

combo

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, listIndex, length, options, selectedIndex

listIndex Property

Gets or sets the list index.

*element.***listIndex** [**=***integer*]

Parts

element
An object expression that evaluates to an intrinsic control.

Integer
Optional. The list index to select. Note that this must be between 0 and ListCount - 1.

Return Values

Returns the index of the currently selected element. If more than one is selected, it returns the first.

Remarks

listIndex is the index of the selected element in the combo. Applies to the combo element.

Applies To

combo

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, length, options, selectedIndex

length Property

Gets the number of options in a select element.

*element.***length**

Parts

element
An object expression that evaluates to a select element.

Return Values

Returns an integer specifying the number of options in a select element.

Applies To

select

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, options, selectedIndex

options Property

Gets the <options> tags for a select element.

*element.***options**

Parts

element
An object expression that evaluates to a select element.

Return Values

Returns an string specifying the <options> tag for a select element.

Remarks

The options array has the following properties:

- defaultSelected
  Identifies the currently selected attribute
- index
  Specifies the index of an option
- length
  Specifies the number of options in the selected object.
- name
  Specifies the name attribute of the selected object.
- selected
  Used to programmatically select an option.
- selectedIndex
  Specifies the index of the selected option.
- text
  Specifies the text to be displayed (this text follows the <option> tag).
- value
  Specifies the value attribute.

Applies To

select

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, selectedIndex

selectedIndex Property

Gets the index for the selected option (or the first option selected when there are multiple selected objects).

element.**selectedIndex**

Parts

element
An object expression that evaluates to a select element.

Return Values

Returns an integer specifying the index for the selected option in a select element.

Applies To

select

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options

click Method

Clicks the element.

*element.***click**

Parts

element
An object expression that evaluates to an intrinsic control.

Applies To

button, **reset**, **submit**, **check box**, **radio**, **combo**

Methods

focus, blur, select, removeItem, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

focus Method

Sets the focus to the element.

*element.***focus**

Parts

element
An object expression that evaluates to an intrinsic control.

Applies To

All elements

Methods

click blur, select, removeItem, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

blur Method

Clears the focus from the element.

*element.***blur**

Parts

element
An object expression that evaluates to an intrinsic control.

Applies To

password, **text**, **text area**

Methods

click, focus, select, removeItem, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

select Method

Selects the contents of the element.

*element.***select**

Parts

element
An object expression that evaluates to an intrinsic control.

Applies To

password, **text**, **text area**

Methods

click, focus, blur, removeItem, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

removeItem Method

Removes the item at index from the element.

*element.***removeItem** *index*

Parts

element
An object expression that evaluates to an intrinsic control.

index
The index of the item to remove. Note that this must be between 0 and ListCount - 1.

Applies To

combo

Methods

click, focus, blur, select, addItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

addItem Method

Adds the item to the element before the item at index.

*element.***addItem** *index*

Parts

element
An object expression that evaluates to an intrinsic control.

index
The index of the item to add. Note that this must be between 0 and ListCount.

Applies To

combo

Methods

click, focus, blur, select, removeItem, clear

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

clear Method

Clears the contents of the element.

*element.***clear** *index*

Parts

element
An object expression that evaluates to an intrinsic control.

index
The index of the item to clear.

Applies To

combo

Methods

click, focus, blur, select, removeItem, addItem

Events

onClick, onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

Events

There are two ways to script events from objects:

1 Using the onEvent="subroutine" syntax. This method can be used for any HTML intrinsic elements, such as forms, buttons, or links. This method does not work for items inserted using the OBJECT tag. The following example uses this syntax in Button1 to handle onClick:

```
<form name="Form1">
     <input type="button" name="Button1" value="Press me"
onClick="pressed">
</form>

<script language="VBScript">
     sub pressed
     alert "I've been pressed"
     document.Form1.Button1.value="OUCH"
     end sub
</script>
```

2 Using the FOR="object" EVENT="eventname" syntax. This method can be used for any named elements, plus any elements inserted using the OBJECT tag. The following example is the same as the first but with a different syntax:

```
<form name="Form1">
     input type="button" name="Button1" value="Press me">
     <script for="Button1" event="onClick" language="VBScript">
     alert "I've been pressed"
     document.Form1.Button1.value="OUCH"
     </script>
</form>
```

onClick Event

Fired when the element is clicked.

*element.***onClick**

Values

element
An object expression that evaluates to an intrinsic control.

Applies To

button, **reset**, **submit**, **check box**, **radio**, **combo**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onFocus, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

onFocus Event

Fired when the element gets the focus.

*element.***onFocus**

Values

element
An object expression that evaluates to an intrinsic control.

Applies To

All elements

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onBlur, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

onBlur Event

Fired when the element loses the focus.

*element.***onBlur**

Values

element
An object expression that evaluates to an intrinsic control.

Applies To

password, **text**, **text area**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onChange, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

onChange Event

Fired when the element has changed.

*element.***onChange**

Values

element
An object expression that evaluates to an intrinsic control.

Applies To

text, **text area**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onSelect

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

onSelect Event

Fired when the contents of the element are selected.

*element.***onSelect**

Values

element
An object expression that evaluates to an intrinsic control.

Applies To

text, **text area**

Methods

click, focus, blur, select, removeItem, addItemclear

Events

onClick, onFocus, onBlur, onChange

Properties

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

history Object

An object that resides below the window in the scripting object model. This object accesses the history list from the browser.

{ewc msdncd, EWGraphic, grpsweeper 14 /a "sweeper.bmp"}

The history object exposes methods for navigating through the current history.

Methods

back, forward, go

Properties

length

length Property

Returns the length of the history list.

*history.***length**

Parts

history
An object expression that evaluates to a history object.

Return Values

Returns the number of entries in the history.

Always returns zero in current implementation.

Applies To

History

Methods

back, forward, go

back Method

Jumps back in the history *n* steps. This behaves exactly as if the user has clicked on the back button *n* times.

*history.***back** *n*

Parts

history
An object expression that evaluates to a history object.

n
The number of pages to jump back in the history. N is always an integer >= 0.

Remarks

Disabled in current implementation.

Applies To

History

Methods

forward, go

Properties

length

forward Method

Jumps forward in the history *n* steps. This behaves exactly as if the user has clicked on the forward button *n* times.

*history.***forward** *n*

Parts

history
An object expression that evaluates to a history object.

n
The number of pages to jump forward in the history. N is always an integer >= 0.

Remarks

Disabled in current implementation.

Applies To

History

Methods

back, go

Properties

length

go Method

Goes to the *n* th item in the history, where *history.***go** 1 jumps to the first item and *history.***go** history.length jumps to the last item.

*history.***go** *n*

Parts

history
An object expression that evaluates to a history object.

n
The index of the history entry, from 1 to history.length.

Remarks

Disabled in current implementation.

Applies To

History

Methods

back, forward

Properties

length

navigator Object

An object that resides below the window in the scripting object model. This object represents "stub" variables for the Netscape Navigator browser. Navigator Object makes your code compatible with the Navigator browser.

{ewc msdncd, EWGraphic, grpsweeper 15 /a "sweeper.bmp"}

The navigator object provides information about the browser application to script writers.

Properties

appCodeName, appName, appVersion, userAgent

appCodeName Property

Returns the code name of the application.

*navigator.***appCodeName**

Parts

navigator
An object expression that evaluates to a history object.

Return Values

Returns a string containing the current application code name.

Applies To

Navigator

Properties

appName, appVersion, userAgent

appName Property

Returns the name of the application. Internet Explorer 3.0 currently returns "Microsoft".

*navigator.***appName**

Parts

navigator
An object expression that evaluates to a history object.

Return Values

Returns a string containing the current application name.

Applies To

Navigator

Properties

appCodeName, appVersion, userAgent

appVersion Property

Returns the version of the application.

*navigator.***appVersion**

Parts

navigator
An object expression that evaluates to a history object.

Return Values

Returns a string containing the current application version.

Applies To

Navigator

Properties

appCodeName, appName, userAgent

userAgent Property

Returns the user agent of the application. Internet Explorer 3.0 currently returns "Mozilla/2.0".

*navigator.***userAgent**

Parts

navigator
An object expression that evaluates to a history object.

Return Values

Returns a string containing the current application user agent.

Applies To

Navigator

Properties

appCodeName, appName, appVersion

Overview

This section explains how to digitally sign files with the Authenticode™ technology included in Microsoft's ActiveX™ SDK. These digital signatures associate a software vendor's name and unique public key with a file, thus assuring some level of accountability.

The section includes:

- An introduction to code signing.
- Procedures for signing code using Authenticode.
- Procedures for checking that the code was signed successfully.
- A glossary.
- Appendices with related information.

Currently, Authenticode allows software vendors to sign:

- .exe files.
- .cab files.
- .ocx files.
- .class files.

Introduction to Code Signing

This section is a general introduction to code signing. Later, we will discuss Microsoft's technology, called Authenticode, which helps developers to easily sign their code.

One of the larger questions facing the software industry is this: How can users trust code that is published on the Internet? Currently, most Web pages contain only static information, but soon they will be filled with controls and applications that are downloaded and run locally, on the user's computer.

Packaged software uses branding and trusted sales outlets to assure users of its integrity, but these are not available when code is transmitted on the Internet. Additionally, there is no guarantee that the code hasn't been altered while being downloaded. Browsers typically exhibit a warning message explaining the possible dangers of downloading data, but do nothing to actually see whether the code is what it claims to be. A more active approach must be taken to make the Internet a reliable medium for distributing software.

Ensuring Integrity and Authenticity

There are two issues that must be addressed:

- Ensuring authenticity, which means assuring users that they know where the code came from.
- Ensuring integrity, which means checking that the code hasn't been tampered with since it was published.

Microsoft's solution to these issues is Authenticode coupled with an infrastructure of trusted entities. We will discuss the infrastructure later in this document, when we explain certification authorities. Authenticode, which is based on industry standards, allows developers to include information about themselves and their code with their programs through the use of *digital signatures*.

Digital Signatures

You use digital signatures when you have data you want to distribute, and you want to assure the recipients that it does indeed come from you. Signing data does not alter it. It simply generates a digital signature string you can bundle with the data.

Digital signatures are created using a public-key signature algorithm such as the RSA public-key cipher. A public-key algorithm actually uses two different keys: the *public key* and the *private key*. (These are called a key pair.) The private key is known only to its owner, while a public key can be available to anyone. Public-key algorithms are designed so that if one key is used for encryption, the other is necessary for decryption. Furthermore, the decryption key cannot reasonably be calculated from the encryption key. In digital signatures, the private key generates the signature, and the corresponding public key validates it.

In practice, public-key algorithms are often too inefficient for signing long documents. To save time, digital signature protocols use a cryptographic digest, which is a one-way hash of the document. The hashed document is signed instead of the document itself. Both the hashing and digital signature algorithms are agreed upon beforehand. Here is a summary of the process:

1. A one-way hash of the document is produced.
2. The hash is encrypted with the private key, thereby signing the document.
3. The document and the signed hash are transmitted.
4. The recipient produces a one-way hash of the document.
5. Using the digital signature algorithm, the recipient decrypts the signed hash with the sender's public key.

If the signed hash matches the recipient's hash, the signature is valid and the code is intact.

When code is associated with a publisher's unique signature, distributing software on the Internet is no longer an anonymous activity. Digital signatures assure accountability, just as a manufacturer's brand name does on packaged software. If an organization or individual wants to use the Internet to distribute software, they should be willing to take responsibility for that software. This approach is based on the premise that accountability is a deterrent to the distribution of harmful code.

Public-Key Certificates

Public keys, because they are public, are vulnerable to attack. In particular, an unscrupulous individual may want to substitute one public key for another.

For example, if Microsoft wants to send an encrypted message to Bob, an independent developer, it goes to the public-key database and gets Bob's public key. Unfortunately, Bob has a rival named Trent. Trent is not only a rival, but a sneak as well. He has substituted his own key for Bob's in the public-key database. Microsoft encrypts a message in Trent's key and sends it to Bob. Trent intercepts the message, decrypts it, and reads it. Finally, he re-encrypts it with Bob's key and sends it on to Bob. Neither Bob nor Microsoft know what has happened.

Public-key certificates are used to prevent this situation. A certificate is someone's public key, signed by a trustworthy person or organization. If Bob has a certificate in the database, it contains much more than his public key. It contains information about Bob, such as his name and address, and it is signed by some entity Microsoft trusts.

Certification Authorities

Trustworthy persons or organizations are called *certification authorities* (CAs). Certificates are verified through a hierarchy of these CAs. Each certificate is linked to the certificate of the CA that signed it. By following this hierarchy, or *verification path*, to a known, trusted CA, you can be assured that a certificate is valid. An example of this is illustrated in the following diagram.

{ewc msdncd, EWGraphic, grpsweeper 16 /a "sweeper.bmp"}

Sample Certification Hierarchy

In this example, Netwerks's certificate is certified by CA1 while Bob's is certified by CA3. Netwerks knows CA1's public key. CA2 has a certificate signed by CA1, so Netwerks can verify the CA2 certificate. The root also has a certificate signed by CA1. CA3 (Bob's CA) has a certificate signed by the root. By moving up the verification chain to a common point (in this case, the root), Netwerks can verify Bob's certificate.

Duties of Certification Authorities

Certification authorities have two main duties:

- They publish the criteria for granting certificates.
- They grant certificates if an applicant meets the published criteria.

Other duties may include:

- Managing certificates (for example, enrolling, renewing, and revoking them).
- Storing root keys.
- Verifying evidence submitted by applicants.
- Providing tools for enrollment.
- Accepting the liability associated with these responsibilities.

Local Registration Agencies

Companies that don't want to take on all the responsibilities associated with being a CA can become a local registration agency (LRA). The LRA views enrollment requests, verifies evidence, and passes on the approved request to the signing CA. The relationship between a CA and an LRA can vary depending on their arrangement, which is specified in a contract.

A possible LRA would be a university acting on behalf of its students. Any university could verify whether a student is actually enrolled and, consequently, could easily approve certificate requests.

Umbrella Organization

It is important that the software industry endorse both the policies that allow CAs to participate, and the criteria that define a responsible commercial or individual software publisher. To ensure this industry cooperation, Microsoft is working to establish an umbrella organization. Over time it is expected that a consortium of industry partners and certificate authorities or an industry organization similar to the Software Publishers Association or the World Wide Web Consortium will fill this role.

Obtaining Certification

To obtain a certificate from a CA, a software publisher must meet the criteria for either a commercial or an individual publishing certificate, and submit these credentials to either a CA or an LRA. The criteria we will discuss are those proposed by Microsoft. Please note that standards bodies such as the World Wide Web Consortium are reviewing them and they are subject to change. We will then describe the overall process of obtaining a certificate for code signing.

Criteria for Commercial Certification

Applicants for a commercial software publishing certificate must meet the following criteria:

- Identification—Applicants must submit their name, address, and other material that proves their identity as a corporate representative. Proof of identify requires either personal presence or registered credentials.
- The pledge—Applicants must pledge that they will not distribute software that they know, or should have known, contains viruses or would otherwise harm the user's computer or code.
- Dun & Bradstreet rating—Applicants must achieve a level of financial standing as indicated by a D-U-N-S® number (which indicates a company's financial stability), and any additional information provided by this service. This rating identifies the applicant as a corporation that is still in business. (Other financial rating services are being investigated.) Corporations that do not have a D-U-N-S number at the time of application (usually because of recent incorporation) can apply for one and expect a response in less than two weeks.
- Private key protection—Applicants must agree to generate and store their private key using a dedicated hardware solution. This can be, for example, a magnetic stripe card, a plastic key with an embedded ROM chip (called a ROM key), or a smart card. For more information about storing keys, see Section 8.7 of Bruce Schneier's book, *Applied Cryptography*.

Two immediate questions are how do large software publishers determine who should apply for certificates and who should sign code? The answer depends on how the software publisher wants to control distribution of software on the Internet.

In a centralized approach, where the company wants total control of what code is published, there may be only one certificate, and strict guidelines for releasing code through one source. Other software publishers may allow each division, or even smaller groups or individuals within the company, to sign their own code using the corporate name. The point is that the software publisher must decide who can apply for a certificate and sign code and who takes responsibility for any code signed using certificates that bear the corporate name.

Using the Dun & Bradstreet rating as a criterion draws a line between "commercial" and "individual" developers. The intended distinction is between commercial persons or entities (that is, sole proprietors, partnerships, corporations, or other organizations that develops software as a business) and non-commercial persons or entities (that is, individuals or nonprofit corporations).

Criteria for Individual Certification

Applicants for an individual software publishing certificate must meet the following criteria:

- Identification—Applicants must submit their name, address, and other material that will be checked against an independent consumer database to validate their credentials.
- The pledge—Applicants must pledge that they cannot and will not distribute software that they know, or should have known, contains viruses or would otherwise maliciously harm the user's computer or code.

The value of an individual software publishing certificate is in the information it provides to users so they can decide whether or not to download the code. Knowing who authored the code, and that the bits have not been altered from the time the code was signed to the present, is reassuring information. Additionally, a browser could provide links to a publisher's Web pages so the user can obtain detailed information about the signed code, the author, and the certificate authority. After learning about this code and the author, the user may decide to run the code, or all future code, coming from this particular individual.

The Application Process

The process of applying for certification is summarized in the following diagram.

{ewc msdncd, EWGraphic, grpsweeper 17 /a "sweeper.bmp"}

These are the steps to apply for and grant a certificate:

1. Apply for a software publishing certificate

In this diagram, a software publisher's request for certification is sent to the local registration agency. (In a simpler model, it is sent to the CA.) It is expected that CAs and LRAs will have Web sites that step the applicant through the application process. Applicants will be able to look at the entire policy and practices statements of the CA or LRA. The utilities an applicant needs to generate signatures, such as Microsoft's Authenticode, should also be available.

The applicant must generate a key pair using either hardware or software encryption technology. The public key is sent to the LRA during the application process. For individuals, all of the necessary information can be transferred on-line. For commercial publishers, because of the identity requirements, proof of identification must be sent by mail or courier.

2. Verify the applicant's credentials

Depending on the contract between the CA and the LRA, these companies will examine the evidence to verify an applicant's credentials. To do this, they may employ external contractors such Dun & Bradstreet.

3. Generate and issue the software publisher X.509 certificate

After the CA has decided that the applicant meets the policy criteria, it generates a Software Publisher Certificate (SPC) that conforms to the industry standard X.509 certificate format with Version 3 extensions. This certificate, which is distributed in the digital signature for the software, identifies the publisher, contains the publisher's public key, and is used to verify that the file has not been modified since it was signed. It is stored by the CA for reference and a copy is returned to the applicant via electronic mail.

The publisher should review the contents of the certificate and verify that the public key works with the private key. After accepting the certificate, the publisher should include a copy in all published software signed with the private key.

Commercial developers can expect a response to their application in less than two weeks. While there is no limit to the number of certificates commercial software publishers can obtain, it is up to the publisher to determine who gets a certificate, and how code is signed and distributed.

4. Distribute signed software

The publisher can now begin signing and distributing software on the Internet. Publishers use utility programs to sign the software they intend to publish. The utility programs use the private key to generate a digital signature on a digest of the binary file and create a signature file containing a PKCS #7 signed-data object. (For more information about PKCS #7, see the RSA specification listed in the "Suggested Reading" section of this document.) The PKCS #7 signed-data object also contains a copy of the X.509 software publisher certificate. For portable executable (PE) image format files, the PKCS #7 signature file contents are stored in the binary file itself, in an additional section.

Signing Code with Authenticode

This section demonstrates how to sign code by creating digital signatures and associating them with files using Authenticode, which is provided with the ActiveX SDK. As we have seen, creating a fully verifiable certificate may assume the existence of a complex hierarchy of CAs. For testing purposes *only*, we provide a root certificate and a root private key. If you are an independent software vendor, you must obtain a certificate from GTE, VeriSign, Inc. or another CA before you begin signing code.

Authenticode

Authenticode consists of programs to digitally sign files and programs to check that the files were, indeed, successfully signed. Before you begin, first check that the underlying CryptoAPI is running. To do this, type:

c:>api *

This should generate SUCCESS messages until its stopped.

The programs are:

- MakeCert, which creates a test X.509 certificate.
- Cert2SPC, which creates a test SPC.
- SignCode, which uses the SPC to sign a file.
- PeSigMgr, which checks to see that the file was signed.
- ChkTrust, which checks the validity of the file.

We will now discuss these programs in more detail.

MakeCert

Use the MakeCert program to generate a test X.509 certificate. The program does the following:

1. It creates a public/private key pair for digital signatures and associates it with a name that you choose.
2. It associates the key pair with a publisher's name that you choose.
3. It creates an X.509 certificate, signed by the root key or one you specifiy, that binds your name to the public part of the key pair. If you do not specify a root key, MakeCert generates one for you.

The syntax for invoking MakeCert is:

MAKECERT [*options*] *outputfile* where the options are:

- -u:subjectKey is the location of the publisher's key pair name. If a key pair does not exist, one is created.
- -k:subjectKeyFile is the location of the publisher's .pvk file.
- -n:name is the name for the publisher's certificate. This name must conform to the X.500 standard. The easiest way to do this is to use the form "CN=*MyName*."
- -d:displayname is the display name of the publisher in the SPC UI.
- -s:issuerKeyFile is the location of the issuer's key. The default is the root key.
- -i:issuerCertFile is the location of the issuer's certificate.
- -b:logoFile is the file name of the SPC agency logo (for example, a .bmp file).
- -l:policyLink is a link to SPC agency policy information (for example, a URL).
- -U:subjectCertFile is the certificate file name with the existing subject public key to be used.
- -#:serialNumber is the serial number of the certificate. The maximum value is 2^31. The default is a value generated by the program that is guaranteed to be unique.
- -I means the certificate will be used by individual software publishers.
- -C means the certificate will be used by commercial software publishers.
- -C:f means the certificate will be used by commercial software publishers who have met the minimum financial criteria.
- -S:session is the session name for an enrollment session.
- -P:purpose is the purpose for which the certificate is being generated. This parameter can either be CodeSigning (this is the default) or ClientAuth.
- -x:providerName is the CryptoAPI provider to use. (See the CryptoAPI documentation for more details.)
- -y:nProviderType is the CryptoAPI provider type to use. (See the CryptoAPI documentation for more details.)
- -K:keyspec is the key specification. This parameter can either be S for a signature key (this is the default) or E for a key-exchange key.
- -B:dateStart is the date when the certificate first becomes valid. The default is when the certificate is created.
- -D:nMonths is the duration of the validity period.
- -E:dateEnd is the date when the validity period ends. The default is the year 2039.
- -h:numChildren is the maximum height of the tree below this certificate.
- -t:types is the certificate type. This parameter can be E for end-entity, C for certificate authority, or both.
- -g creates a glue certificate.

- -r creates a self-signed certificate.
- -m means to use the MD5 hash algorithm. This is the default.
- -a means to use the SHA1 hash algorithm.
- -? displays the options.

Here is an example:

>MakeCert -u:MyKey -n:CN=MySoftwareCompany Cert.cer

This generates a certificate file called Cert.cer. The public part of the key pair called KeyName is bound to the publisher, MySoftwareCompany.

This utility program should not be used once the software publisher obtains a valid X.509 software publisher certificate from the appropriate CA.

Cert2SPC

After you have generated a certificate, you must create an SPC with the Cert2SPC program. This program wraps the X.509 certificate and the root certificate into a PKCS#7 signed-data object. PKCS#7 objects are commonly used to carry certificates because it is possible to put several of them into a single object. Again, this program is for test purposes only. A valid SPC is obtained from a CA.

The syntax for Cert2SPC is:

Cert2SPC *cert1*.cer *cert2*.cer .... certN.cer *output*.spc

where:

- *cert1...certN* are the names of the X.509 certificates.
- *output* is the name of the SPC. This is a PKCS#7object containing the X.509 certificate and the root certificate.

Here is an example:

>Cert2Spc root.cer cert.cer cert.spc

This combines Cert.cer and Root.cer to make an SPC called Cert.spc.

SignCode

The final step is to use the SPC to actually sign a file. This is done with the SignCode program. This program will:

1. Create a cryptographic digest of the file.
2. Sign the digest with your private key.
3. Extract the X.509 certificates from the SPC.
4. Create a new PKCS#7 signed-data object that contains the serial numbers of the certificates and the signed digest information.
5. Embed the object into the file.

If you have a valid SPC, then you can use this program to actually sign your code. The SignCode program has a wizard to help you do this. To sign code using the wizard, simply type SignCode, without any options. If you want to sign your code manually, the syntax is:

SignCode [-prog *filename* -spc *credentials* -pvk *privateKeyFile*

[-name *opusName* [-info *opusInfo*]]] [-gui] [-nocerts]

where:

- -prog *filename* is the name of the file to sign.
- -spc *credentials* is the file that contains the credentials. This is usually an .spc file.
- -pvk *privateKeyFile* is the file containing the private key of the publisher. This is usually a .pvk file.
- -name *opusName* is a name for your program.
- -info *opusInfo* is a location, such as an URL, for obtaining information about your program.
- -gui invokes the wizard
- -nocerts means you don't want any X.509 certificates embedded in the PKCS#7 signed-data object. In this case, the relevant certificates must already be stored on the client computer.
- -provider *providerName* is the CryptoAPI provider to use. (See the CryptoAPI documentation for more details.)
- -providerType *providerType* is the CryptoAPI provider type to use. (See the CryptoAPI documentation for more details.)
- -commerical means the code being signed was created by a commercial software publisher.
- -individual means the code being signed was created by an individual software publisher.
- -sha means you want to use the SHA hashing algorithm.
- -md5 means you want to use the MD5 hashing algorithm. This is the default hashing algorithm.
- -? displays the options.

Here is an example of how to sign a file:

>SignCode -prog MyProgram.exe -spc cert.spc -pvk MyKey

This embeds a PKCS#7 object, Cert.spc, into the digest of file, MyProgram. The digest is signed with the private key of the MyKey key pair.

Once this is done (assuming you have a valid certificate), the file can be distributed to your customers.

PeSigMgr

The PeSigMgr program checks to see if SignCode was successful. This means the file should have a PKCS#7 object embedded in it. Here is the syntax:

PESIGMGR [*options*] *signedfile*

where:

- -l lists the certificates in an image.
- -a:<*filename* > adds a certificate file to an image
- -r:<*index* > removes certificate <*index* > from an image.
- -s:<*filename* > is used with -r to save the removed certificate.
- -t:<*CertType* > is used with -a to specify the type of certificate, where CertType may be X509 or PKCS7. (The default is PKCS7).
- -? displays the options.
- *signedfile* is the name of the signed file you want to check.

Here is an example:

>PeSigMgr -l MyProgram.exe

A sample response is:

>Certificate 0 Revision 256 Type PKCS#7

This means a certificate was embedded in the file.

ChkTrust

The ChkTrust program checks the validity of the file. It does this by:

1. Extracting the PKCS#7 signed-data object
2. Extracting the X.509 certificates from the PKCS#7 signed-data object.
3. Computing a new hash of the file and comparing it with the signed hash in the PKCS#7 object.

If the hashes agree, ChkTrust then verifies that the signer's X.509 certificate points back to the root certificate and that the correct root key was used.

If all these steps are successful, the file has not been tampered with, and the vendor was authorized to publish the file by the root authority.

Here is the syntax:

CHKTRUST [*type*] *signedfile*

where:

- -c is a cabinet file.
- -i is a PE image file.
- -j is a Java class file.

Here is an example:

ChkTrust MyProgram.exe

A successful response is:

Result: 0

Glossary

X.509 Certificate

A cryptographic certificate that contains a vendor's unique name and the vendor's public key.

PKCS#7 Signed Data

A Public Key Certificate Standard #7 (PKCS#7) signed-data object encapsulates the information used to sign an file. Typically, it includes the signer's certificate, the root certificate, and the signer's public key.

Certification Authority (CA)

A trusted entity that makes a statement (represented by an X.509 certificate) about the authenticity of another certificate.

Cryptographic Digest

A one-way hash function that takes a variable-length input string and converts it to a fixed-length output string (called a cryptographic digest). This fixed-length string "fingerprints" the file by producing a value that indicates whether a file submitted for download matches the original file.

Local Registration Authority (LRA)

An intermediary between a publisher and a CA. The LRA can, for example, verify a publisher's credentials before sending them to the CA.

Portable Executable (PE) Image

The standard Win32® executable format.

Software Publishing Certificate (SPC)

A PKCS#7 signed-data object containing X.509 certificates, and public key signatures.

Trust Provider

The portion of the operating system that decides whether or not a given file is trusted. This decision is based on the certificate associated with the file.

WIN_CERTIFICATE

A Win32 data structure that contains either a PKCS#7 signed-data object or an X.509 certificate.

Appendix A - Required Files

To use Authenticode, the following files are required:

Wintrust.dll (installed in the System\System32 directory)

Digsig.dll (installed in the System\System32 directory)

Makecert.exe (creates an X.509 certificate for testing purposes only)

Cert2SPC.exe (creates an SPC for testing purposes only)

SignCode.exe (digitally signs code)

PeSigMgr.exe (checks that SignCode was successful)

ChkTrust.exe (checks the validity of the file)

Root.cer (a root certificate for testing purposes only)

Appendix B - The X.509 Certificate

The X.509 protocols include a structure for public-key certificates. A CA assigns a unique name to each user and issues a signed certificate containing this name and the user's public key. The following diagram shows an X.509 certificate.

{ewc msdncd, EWGraphic, grpsweeper 18 /a "sweeper.bmp"}

X.509 Certificate

These are the meanings for each field.

Field	Meaning
Version	Identifies the certificate format.
Serial Number	Is unique to the CA.
Algorithm Identifier	Identifies the algorithm used to sign the certificate, together with any necessary parameters.
Issuer	The name of the CA.
Period of Validity	A pair of dates. The certificate is valid during the time period between the two.
Subject	The name of the user.
Subject's Public Key	Contains the public key algorithm name, any necessary parameters, and the public key.
Signature	The CA's signature.

Appendix C - Suggested Reading

The topic of digital signing is discussed more fully in the following documents:

CCITT, Recommendation X.509, *The Directory-Authentication Framework*, Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.

Microsoft Cryptographic Service Provider Programmer's Guide, Microsoft, 1995.

Microsoft Application Programmer's Guide, Microsoft, 1995

RSA Laboratories, *PKCS#7: Cryptographic Message Syntax Standard*. Version 1.5, November, 1993.

Schneier, Bruce. *Applied Cryptography*, 2d ed. New York: John Wiley & Sons, 1996.

http://www.microsoft.com/devonly/

http://rsa.com

WebPost Overview

The components of the WebPost Software Development Kit (SDK) allow authoring tools to easily post Web pages (files) to the user's Internet Web site. The WebPost functions can be used to connect to the Internet Service Provider (ISP), determine the protocol needed to copy the files, and so on. Optionally, these functions can also display a wizard to guide the user through posting a file.

WebPost API

The WebPost application programming interface (API) makes it possible for authoring tools to post Web pages to an Internet site with just one call to the WpPost function. In a typical scenario, an authoring tool's **File** menu would include a **Post to Web** button that, when clicked, causes the tool to call the WpPost function. This function starts a wizard that asks the user for a friendly name for the Internet site, the Uniform Resource Locator (URL) for the given site, and the name of a dial-up connection for accessing the Internet. Next, the function connects to the Internet server at the given URL, determines the protocol to use for posting the Web pages, and then posts the requested files. For subsequent posting to the same site, this function remembers the details of how to connect to the site, and posts the files with little user intervention. Thus, the WebPost API maintains an association between a friendly site name, remembers all the details involved in posting to that site, and allows the applications to easily post Web pages to a site or to the URL associated with that site.

As an alternative to the wizard, an authoring tool can let the user choose the Web site before calling WpPost by displaying a list of sites (plus a New Site item) obtained by a call to the WpListSites function. The list of sites could be in a nested menu that appears when the user clicks the **Post to Web** button.

WebPost Service Providers

The WebPost dynamic-link library (DLL) can post Web pages to some of the popular types of Internet servers, including the National Center for Supercomputing Applications' (NCSA's) *httpd* and Microsoft's Internet Information Server (IIS). To post to other types of Internet servers, the WebPost API uses the WebPost Service Provider Interface (SPI) to communicate with DLLs that "know how" to post Web pages to those servers.

{ewc msdncd, EWGraphic, grpsweeper 19 /a "sweeper.bmp"}

A WebPost service provider is implemented as an OLE Component Object Model (COM) server. An authoring tool can take advantage of the functionality available from a WebPost service provider by calling functions implemented by the provider. An authoring tool uses the WpBindToSite function to retrieve the addresses of the provider's functions. For descriptions of the functions that a provider implements, see WebPost SPI Interface Functions.

Posting Information File

Because of variations among Internet servers, Microsoft requests that Internet Service Providers (ISPs) include a posting information file named postinfo.html on the root URLs of their Internet servers. (Because file names are case-sensitive on some servers, the name of the postinfo.html file should be in all lowercase letters.) This file contains details about the posting protocol and policy. The standard WebPost service provider uses the information in the file to help with the detection of the WebPost protocol.

The following is a sample postinfo.html file that the default provider uses. The comments section that follows the <body> tag contains configuration information for authoring tools that use the Microsoft WebPost functions.

```
<html>
<!-- postinfo.html version 0.100 -->

<head>
<title>
    Web Posting Information
</title>
<head>

<body>
<!--
    WebPost
    version="0.100"
    BaseURL="http://<servername>/~$USERNAME"
    BasePath="public_html"
    FtpServerName="<servername>"
    XferType="FTP"
    DefaultPage="default.htm"
    VerifyFiles="1"
    CreateRoot="1"
-->

<h1>
    Web Posting Information
</h1>
```

Provider DLL Installation

Each provider should create a subkey in the following registry location:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
    \WebPost\Providers
```

The subkey consists of the WpSite class identifier (CLSID) that the provider exports. Within this subkey, you should enter the following values:

Value	Meaning
Description	A string that contains the name of the provider. This name is used in the wizard; it appears in the list box that contains the names of the providers.
Path	The path of the WebPost service provider DLL.
Priority	A doubleword value that indicates the provider's priority. A value of 0 is high priority. The default provider has a priority of 8192.

The following example shows the registry entries for the default provider:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Providers
    \{3151E2E0-6C4C-11CF-86B1-00AA0060F86C}
        "Provider"="Other Internet Provider"
        "Path"="defwpp.dll"
        "Priority"=hex:00,10,00,00
```

All providers should also register their class identifiers at the following registry location, with the value name set to the class identifier. WebPost functions use the entries in this registry location to determine which providers to load.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\WebPost\Providers
```

The default provider entry at this location is as follows:

```
"{3151E2E0-6C4C-11CF-86B1-00AA0060F86C}"=""
```

WebPost API Functions

This section describes the functions in the WebPost API.

The functions for simple usage include:

WpDeleteSite	Deletes a friendly site name that has been configured.
WpListSites	Retrieves a list of the friendly site names that the user has configured.
WpPost	Posts a file name to the URL at a given site. If the URL and site name are set to NULL, the function starts a wizard that asks the user to choose or set up a URL and site.

The function for advanced usage is:

WpBindToSite	Returns a COM object to the WebPost service provider that supports the given site name or URL. With this object, the application can call into the provider functions listed in WebPost SPI Functions.

A fifth WebPost API function, WpPostFile, is OLE Automation-enabled and very similar to WpPost. The major difference is that WpPostFile can post only one file or directory at a time.

WpBindToSite

Retrieves the address of a WebPost service provider object that can post Web pages to the specified site name or URL. The application can then call the provider functions directly. The function works only if the site exists.

LONG WpBindToSite(
IN HWND *hwnd*,
IN LPTSTR *lpszSiteName*,
IN LPTSTR lpszURL *OPTIONAL*,
IN DWORD *fdwFlags*,
IN DWORD *dwReserved*,
OUT LPVOID **ppbObj* **);**

Parameters

hwnd
Handle of the window to which the focus returns when the wizard, if invoked, completes. This parameter can be NULL if the wizard is not invoked or if this function is called from a console application.

lpszSiteName
Address of a null-terminated string that contains the site name.

lpszURL
Address of a null-terminated string that contains a URL.

fdwFlags
Action flags. Reserved for future use.

dwReserved
Reserved; must be set to zero.

ppbObj
Address of a WebPost service provider interface.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WpDeleteSite

Deletes a site name that has been configured.

LONG WpDeleteSite(
IN LPTSTR *lpszSiteName* **);**

Parameters

lpszSiteName
Address of a null-terminated string that contains the site name.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WpListSites

Retrieves information about the Internet sites that are already configured.

LONG WpListSites(
IN OUT LPDWORD *lpcbSites***,**
OUT LPWPSITEINFO *lpbSites***,**
OUT LPDWORD *lpcSites* **);**

Parameters

lpcbSites
Address of a variable that contains the size, in bytes, of the buffer pointed to by the *lpbSites* parameter. When this function returns, the variable contains the number of bytes in the *lpbSites* buffer. If the *lpbSites* value is NULL, the variable receives the size of the buffer required to contain all of the site structures.

lpbSites
Address of a buffer that receives an array of WPSITEINFO structures that contain the site information.

lpcSites
Address of a variable that receives the number of structures returned in the *lpbSites* array. If the *lpbSites* value is NULL, the variable receives the total number of sites.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WpPost

Posts a list of files or directories to an Internet site identified by the given site name or URL. If the site name and URL are NULL, the function invokes a wizard to let the user choose an existing or create a new Internet site.

LONG WpPost(
IN HWND hwnd *OPTIONAL*,
IN DWORD *cLocalPaths*,
IN LPTSTR **lppszLocalPaths*,
IN OUT LPDWORD *lpcbSiteName*,
IN OUT LPTSTR lpszSiteName *OPTIONAL*,
IN OUT LPDWORD *lpcbURL*,
IN OUT LPTSTR lpszURL *OPTIONAL*,
IN DWORD *fdwFlags* **);**

Parameters

hwnd
Handle of the window to which the focus returns when the wizard, if invoked, completes. This parameter can be NULL if the wizard is not invoked or if this function is called from a console application.

cLocalPaths
Number of elements in the array specified by the *lppszLocalPaths* parameter.

lppszLocalPaths
Address of an array of null-terminated strings that contain the file names or directories to be posted on the Internet. If any of these strings point to a directory (and the **WPF_NO_RECURSIVE_POST** flag is not set in the *fdwFlags* parameter), all the files in that directory are posted.

lpcbSiteName
Address of a variable that contains the length, in bytes, of the buffer specified by the *lpszSiteName* parameter.

lpszSiteName
Address of a null-terminated string that contains the friendly Internet site name. If this parameter and the *lpszURL* parameter are NULL, the function invokes the wizard (unless **WPF_NO_WIZARD** is set in *fdwFlags*) to let the user choose or create a site name.

If this parameter is not NULL, on return it will contain the site name that the files were posted to.

lpcbURL
Address of a variable that receives the length, in bytes, of the buffer specified by *lpszURL*.

lpszURL
Address of a null-terminated string that contains the destination URL. If this parameter is NULL, the files are posted in the root URL for *lpszSiteName*.

If this parameter is not NULL, the URL of the file copied, or the URL of the directory that the files were copied to, is returned in the buffer pointed to by this parameter.

fdwFlags
Array of action flags. The following values can be combined:

Value	Meaning
WPF_FIRST_FILE_AS_DEFAULT	Take the first file specified in the *lppszLocalPaths* list as the file that will be shown as the default page.
WPF_MINIMAL_UI	Skip the pages where the input has been provided. For example, if

	lpszSiteName is specified, the wizard will not show the page for choosing the site name.
WPF_NO_RECURSIVE_POST	If any element in the *lppszLocalPaths* array points to a directory, do not post files recursively.
WPF_NO_WIZARD	Do not prompt the user for any input. This is relevant only if *lpszSiteName* has been created before.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

Remarks

Suppose you have created a site named "My Web Site" that has a URL of "http://www.isp.com/~username", and you want to post a file named "abcd.tmp" to "http://www.isp.com/~username/personal" as person.htm. The parameters for **WpPost** should be as follows:

```
lpszSiteName = "My Web Site"
cLocalPaths = 1
lppszLocalPaths = {"c:\tmp\abcd.tmp"}
lpszURL = "http://www.isp.com/~username/personal/person.htm"
```

WpPostFile

This function is for use with OLE Automation. It posts a file or directory to an Internet site identified by the given site name or URL. If the site name and URL are NULL, the function invokes a wizard to let the user choose an existing or create a new Internet site.

HRESULT WpPostFile(
[in] long *hwnd*,
[in] BSTR *lpszLocalPaths*,
[in,
out]long * *lpcbSiteName*,
[in,
out]BSTR * *lpszSiteName*,
[in,
out]long * *lpcbURL*,
[in,
out]BSTR * *lpszURL*,
[in] long *fdwFlags*,
[out] long **lpRetCode* **);**

Parameters

hwnd
Handle of the window to which the focus returns when the wizard, if invoked, completes. This parameter can be NULL if the wizard is not invoked or if this function is called from a console application.

lpszLocalPaths
Address of a null-terminated string that contains the file name or directory to be posted on the Internet.

lpcbSiteName
Address of a variable that contains the length, in bytes, of the buffer specified by the *lpszSiteName* parameter.

lpszSiteName
Address of a null-terminated string that contains the friendly Internet site name. If this parameter and the *lpszURL* parameter are NULL, the function invokes the wizard (unless **WPF_NO_WIZARD** is set in *fdwFlags*) to let the user choose or create a site name.

If this parameter is not NULL, on return it will contain the site name that the files were posted to.

lpcbURL
Address of a variable that receives the length, in bytes, of the buffer specified by *lpszURL*.

lpszURL
Address of a null-terminated string that contains the destination URL. If this parameter is NULL, the files are posted in the root URL for *lpszSiteName*.

If this parameter is not NULL, the URL of the file copied, or the URL of the directory that the files were copied to, returns in the buffer pointed to by this parameter.

fdwFlags
Array of action flags. The following values can be combined:

Value	Meaning
WPF_FIRST_FILE_AS_DEFAULT	Take the first file specified in *lpszLocalPaths* as the file that will be shown as the default page.
WPF_MINIMAL_UI	Skip the pages where input has been provided. For example, when

	this flag is set and *lpszSiteName* is specified, the wizard will not show the page for choosing the site name.
WPF_NO_RECURSIVE_POST	If *lpszLocalPaths* points to a directory, do not post files recursively.
WPF_NO_WIZARD	Do not prompt the user for any input. This is relevant only if *lpszSiteName* points to a site that has been created before.

lpRetCode
Address of a variable that receives the function's return code.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

Remarks

This function can post only one file or directory at a time.

Here are two examples that show how to use this function with Visual Basic®.

Example 1:

```
Dim wpo as New WPObj.WPObj
wpo.WpPostFile  <etc.>
```

Example 2:

```
Dim wpo as Object
Set wpo = CreateObject("WPObj.Application")
wpo.WpPostFile  <etc.>
```

WPSITEINFO

Contains information about a site on the World Wide Web.

```
typedef struct tagWPSITEINFO {
    DWORD  dwSize;
    DWORD  fdwFlags;
    LPTSTR lpszSiteName;
    LPTSTR lpszSiteURL;
} WPSITEINFO, *LPWPSITEINFO;
```

Members

dwSize
Size, in bytes, of the structure.

fdwFlags
Array of flags that indicate the state of the site and provide information about the site. The following values can be combined:

Value	Meaning
WPSF_NEEDS_COMMIT	Posting occurs during the Commit function.
WPSF_CAN_BROWSE_DIR	Browsing directories is supported.
WPSF_CONNECTED_TO_NETWORK	Connected to the network.
WPSF_LOGGED_IN_TO_SERVER	Logged in to the server.

lpszSiteName
Address of a null-terminated string that specifies a friendly name for the URL pointed to by *lpszSiteURL*.

lpszSiteURL
Address of a null-terminated string that specifies the root URL for this site.

WebPost SPI Functions

WebPost service provider DLLs are OLE COM servers that do the actual work of posting files to Internet sites. They provide the functions described in this section. The WebPost functions, and the wizard, route the request to one of the providers based on the site name or URL and use the provider functions to post the files.

Each provider DLL exports the WppBindToSite function, which takes a site name or URL and an interface identifier, and returns a pointer to the interface. The WpBindToSite function uses the WppBindToSite function to obtain the address of an interface, and then passes the address back to the caller. Each provider also exports the WppListSites function, which lists all the Internet sites that the provider can post to.

WppBindToSite

Retrieves the address of an interface if the provider DLL owns the site name or the URL.

LONG WppBindToSite(
IN HWND *hwnd*,
IN LPCTSTR lpszSiteName *OPTIONAL*,
IN LPCTSTR lpszURL *OPTIONAL*,
IN REFIID *riid*,
IN DWORD *fdwFlags*,
IN DWORD *dwReserved*,
OUT PVOID **ppvObj* **);**

Parameters

hwnd
Handle of the window to which the focus returns when the wizard, if invoked, completes. This parameter can be NULL if the wizard is not invoked or if this function is called from a console application.

lpszSiteName
Address of a null-terminated string that contains an Internet site name.

lpszURL
Address of a null-terminated string that contains a URL.

riid
Interface identifier. All providers should support the IID_IWPSite interface identifier.

fdwFlags
Action flag. Can be the following value:

Value	Meaning
WPF_FORCE_BIND	The provider should return successfully for the function no matter what the *lpszURL* value is.

dwReserved
Reserved; must be set to zero.

ppvObj
Address of a variable to receive the interface address. For more information about the interfaces, see WebPost SPI Interface Functions.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WppDeleteSite

Deletes a site name that has been configured.

LONG WppDeleteSite(
IN LPTSTR *lpszSiteName* **);**

Parameters

lpszSiteName
Address of a null-terminated string that contains the site name.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WppListSites

Retrieves information about the Internet sites managed by this provider. The WpListSites function calls this function for each provider, aggregates the results, and returns them to the caller.

LONG WppListSites(
IN OUT LPDWORD *lpcbSites***,**
OUT LPWPSITEINFO *lpbSites***,**
OUT LPDWORD *lpcSites* **);**

Parameters

lpcbSites
Address of a variable that receives the number of bytes copied to the buffer pointed to by the *lpbSites* parameter. If *lpbSites* is NULL, the variable receives the required size, in bytes, of the buffer.

lpbSites
Address of a buffer that receives an array of WPSITEINFO structures that contain the site information

lpcSites
Address of a variable that receives the number of structures copied to the *lpbSites* buffer. If *lpbSites* is NULL, the variable receives the total number of sites.

Return Values

Returns ERROR_SUCCESS if successful or an error value otherwise.

WebPost SPI Interface Functions

A WebPost SPI provides the interface functions described in this section. There should be an ANSI version and a Unicode version of each interface. Please refer to the Wpspi.h file for details.

AddWizardPages

Allows the provider DLLs to plug pages into the wizard invoked by the WpPost function.

AddWizardPages(
IN LPVOID *lpv*,
IN LPFNADDPROPSHEETPAGE *lpfnAdd*,
IN OUT LPARAM *lParam* **);**

Parameters

lpv
Not currently used.

lpfnAdd
Address of a function that this WebPost service provider should use to add wizard pages.

lParam
Address of a page identifier, an unsigned integer.

As input to this function, the low-order word is the dialog identifier of the next page of the last page of the provider, and the high-order word is the dialog identifier of the previous page of the first page of the provider. If the high-order word is zero, the provider's first page is the first page of the wizard (for the WPF_MINIMAL_UI case). The **Back** button should be disabled using **PropSheet_SetWizButtons**. If the low-order word is zero, the provider's last page is the last page of the wizard. The **Next** button should be changed to the **Finish** button using **PropSheet_SetWizButtons**.

As output from this function, the low-order word is the dialog identifier of the provider's last page, and the high-order word is the dialog identifier of the provider's first page. If the provider has no pages, they should return zero as the page identifier.

Remarks

As an example, in setting up a new site the wizard would provide the friendly site name and the site's URL, and the provider DLLs would handle the rest of the site setup. The dialog identifier of the provider wizard page should be a value between the value of IDD_WEBPOST_PROVIDER_FIRST and IDD_WEBPOST_PROVIDER_LAST.

Commit

Ensures that all the files posted to this server with the PostFiles function are actually written to the Internet server.

Commit(void);

DeleteFile

Deletes the given file from the destination site.

DeleteFile(
IN LPCTSTR *lpszFile*
);

Parameters

lpszFile
Address of a null-terminated string that contains the name of the file to delete.

FindClose

Closes the specified search handle.

FindClose(
IN HANDLE *hSearchHandle*
);

Parameters

hSearchHandle
Search handle returned by a previous call to the FindFirstFile function.

Remarks

The FindFirstFile and FindNextFile functions use the search handle to locate files with names that match a given name.

FindFirstFile

Searches a directory for a file whose name matches the specified file name on the destination site identified by this object. It examines subdirectory names as well as file names.

FindFirstFile(
IN LPCTSTR *lpszSearchFile*,
OUT LPWIN32_FIND_DATA *lpFindFileData*,
OUT LPHANDLE *lpSearchHandle* **);**

Parameters

lpszSearchFile
Address of a null-terminated string that contains the file name to find.

lpFindFileData
Address of a **WIN32_FIND_DATA** structure that receives information about the file or subdirectory that has been found.

lpSearchHandle
Address of a variable that receives a handle that can be used for subsequent calls to the FindNextFile and FindClose functions.

FindNextFile

Continues a file search from a previous call to the FindFirstFile function.

FindNextFile(
IN HANDLE *hSearchHandle*,
OUT LPWIN32_FIND_DATA *lpFindFileData*,**);**

Parameters

hSearchHandle
Search handle returned by a previous call to the FindFirstFile function.

lpFindFileData
Address of a **WIN32_FIND_DATA** structure that receives information about the file or subdirectory that has been found.

GetError

Retrieves additional information about an error.

GetError(
OUT LPDWORD *lpdwErrorType*,
OUT LPDWORD *lpdwErrorCode*,
IN OUT LPDWORD *lpcbError*,
OUT LPTSTR *lpszError* **);**

Parameters

lpdwErrorType
Indicates the error type. To be defined.

lpdwErrorCode
Indicates the error code.

lpcbError
Indicates the size, in bytes, of the *lpszError* buffer.

lpszError
Address of a buffer that receives the error information.

GetParam

Retrieves the configuration parameters for this site.

GetParam(
IN LPCTSTR *lpszParameter***,**
IN OUT LPDWORD *lpcbValue***,**
OUT LPTSTR *lpszValue* **);**

Parameters

lpszParameter
Address of a null-terminated string that contains the configuration parameter name, such as **WPCP_HOMEPAGE_URL**. For more information about configuration parameters, see Configuration Parameters.

lpcbValue
Address of a variable that receives the size of the *lpszValue* buffer. If *lpszValue* is NULL, or if the buffer length is short, this variable receives the size, in bytes, of the parameter name.

lpszValue
Address of a buffer that receives the value of the parameter queried.

GetSiteInfo

Retrieves the site information for the current object.

GetSiteInfo(
OUT LPWPSITEINFO *lpbSite*,
IN OUT LPDWORD *lpcbSite* **);**

Parameters

lpbSite
Address of a buffer that receives a WPSITEINFO structure containing the site information.

lpcbSite
Address of a variable that contains the size, in bytes, of the *lpbSite* buffer.

NetworkConnect

Connects to the Internet. If the site uses a dial-up connection, this function starts the dial-up connection (configured during the site setup).

NetworkConnect(
IN LPCTSTR *lpszUserName***,**
IN LPCTSTR *lpszPassword* **);**

Parameters

lpszUserName
Address of a null-terminated string that contains the user name for the dial-up connection. If this parameter is NULL, the user name created during the setup of this site is reused.

lpszPassword
Address of a null-terminated string that contains the password for the dial-up connection. If this parameter is NULL, the behavior depends on the provider.

NetworkDisconnect

Disconnects from the Internet (hang-up if a dial-up connection was used).

NetworkDisconnect(void);

PostFiles

Posts files to the specified URL on the destination site identified by this object.

PostFiles(
IN DWORD *cLocalPaths*,
IN LPTSTR **lppszLocalPaths*,
IN OUT LPDWORD *lpcbURL*,
IN OUT LPTSTR lpszURL *OPTIONAL*,
IN DWORD *fdwFlags* **);**

Parameters

cLocalPaths
Number of elements in the *lppszLocalPaths* array.

lppszLocalPaths
Address of an array of null-terminated strings that contain the file names or directories to be posted on the Internet. If any of these strings point to a directory and the **WPF_NO_RECURSIVE_POST** flag is not set in *fdwFlags*, all the files in that directory are posted.

lpcbURL
Address of a variable that indicates the length of the *lpszURL* buffer. When the function returns, the variable contains the length, in bytes, of that buffer.

lpszURL
Address of a null-terminated string that contains the destination URL. If this parameter is NULL, the files are posted in the root URL for the site name specified by *lpszSiteName*.

If this parameter is not NULL, the URL of the copied file, or the URL of the directory that the files were copied to, is returned in the buffer pointed to by this parameter.

fdwFlags
Action flags. The following values can be combined:

Value	Meaning
0	Post files, invoking the wizard if necessary.
WPF_FIRST_FILE_AS_DEFAULT	Take the first file specified in the *lppszLocalPaths* list as the file that will be shown as the default page.
WPF_MINIMAL_UI	Skip the introduction page in the wizard.
WPF_NO_RECURSIVE_POST	If any element in the *lppszLocalPaths* array points to a directory, do not post files recursively.
WPF_NO_WIZARD	Do not prompt the user for any input.

Remarks

For more information, see the description of WpPost.

ServerLogin

Logs the user on to the Internet server. The given user name and password are for the Internet server. They may or may not be the same as those used for the NetworkConnect function.

ServerLogin(
IN LPCTSTR *lpszUserName*,
IN LPCTSTR *lpszPassword* **);**

Parameters

lpszUserName
Address of a null-terminated string that contains the user name for the Internet server. If this parameter is NULL, the user name created during the setup of this site is reused.

lpszPassword
Address of a null-terminated string that contains the password for the Internet server. If this parameter is NULL, the behavior depends on the provider.

ServerLogout

Logs the user out of the Internet server.

ServerLogout(void);

SetParam

Sets a configuration parameter for a given site.

SetParam(
IN LPCTSTR *lpszParameter*,
IN LPCTSTR *lpszValue* **);**

Parameters

lpszParameter
Address of a null-terminated string that contains a configuration parameter name, such as **WPCP_DEFAULT_URL**. For more information about configuration parameters, see Configuration Parameters.

lpszValue
Address of a null-terminated string that contains the configuration parameter value.

Configuration Parameters

Every WebPost service provider must support the following parameters:

```
#define WPCP_HOMEPAGE_URL "HomepageURL"
```

A query of this parameter should return the home page for the given site. For example, for www.provider.com running NCSA's httpd server, the home page will usually be *"http://www.provider.com/~username/index.htm"*.

Here are the parameters supported by the default service provider (Defwpp.dll):

```
#define WPCP_BASEURL    "BaseURL"
```

A query of this parameter should return the root URL for the given user. For example, for www.provider.com running NCSA's httpd server, it will usually be "http://www.provider.com/~username".

```
#define WPCP_BASEPATH    "BasePath"
```

A query of this parameter should return the corresponding file path for the above WPCP_BASEURL parameter. For example, for www.provider.com running NCSA's httpd server, it will usually be "public_html".

More parameters are to be defined at a later date.

Beta Release Notes

The WebPost SDK files include Wpapi.h, Wpspi.h, Wpguid.h, Webpost.lib, and this document, Webpost.doc. Wpobj.h and Wpobj.tlb are for OLE Automation programmers.

The WebPost executable files in the SDK CD include Wpwiz.exe, Webpost.dll, and Defwpp.dll. The Wpwiz.exe file is a simple utility that calls the WpPost function. The Webpost.dll file provides the WebPost API functions. The Defwpp.dll file provides the Service Provider Interface (SPI) for the default provider that can post files to NCSA's httpd server. By default, it posts files using the *ftp* protocol to the *public_html* subdirectory under the user's home directory and verifies them with the *http* protocol using the *wininet* interface. It also lets the user identify a dial-up connection to the Web site and can connect to the Internet server over this dial-up connection.

Purpose and Background

The Microsoft Win32 Internet functions provide Win32 applications with easy access to common Internet protocols. These functions abstract the Internet's Gopher, File Transfer Protocol (FTP), and HyperText Transfer Protocol (HTTP) protocols into a high-level application programming interface (API) that is familiar to Win32 independent software vendors (ISVs) and software developers, and which provides a fast and straightforward path to making applications Internet-aware.

NOTE Initially, the Win32 Internet functions will be shipped as redistributables independent of operating systems through the Microsoft Developer Network (MSDN), CompuServe, and the Internet. ISVs can redistribute WININET.DLL with their applications, following the model of Win32s. In the future, the functions described in this documentation will be folded into all Microsoft operating systems. The Win32 Internet functions are exported from the WININET.DLL.

The Win32 Internet functions facilitate access to the Internet in the following ways:

- Eliminates the need to embed knowledge of TCP/IP and Windows Sockets.

  By converting the Internet protocols into task-oriented functions, application developers do not need to write Windows Sockets code or be familiar with the TCP/IP protocol. One or several Windows Sockets functions can be executed by each call to an Internet function.
- Eliminates the need to embed knowledge of Internet protocols.

  While the concepts supported by the Internet protocols such as FTP and HTTP are simple, the actual implementation of these protocols can be complex. For example, FTP servers return ASCII text file directory listings, but parsing these listings requires specific knowledge of the format returned by each FTP server. By encapsulating this functionality within the Internet functions, directory parsing is solved once for all applications using the FTP protocol. This provides consistent behavior across applications.
- Provides a constant set of functions in an environment of rapidly changing and evolving protocols.

  Keeping pace with the changes in Internet protocols is a challenge when writing applications. With the set of functions designed to remain constant, application developers no longer need to update their applications every time the underlying protocol changes. Now, only the WININET.DLL needs to be changed. In addition, advanced protocol features, such as firewall-friendly FTP and Gopher+, can also be implemented without changing applications.
- Follows Win32 function standards.

  The Win32 Internet functions are similar to the traditional Win32 functions in the way they treat elements such as buffer management and error returns. Application developers familiar with the Win32 function set will find that the Win32 Internet functions return information in a familiar format. Furthermore, application developers will find it easy to use the returned information in other Win32 functions.
- Provides both ANSI and Unicode versions of functions.

  Internet protocols use Latin 1 character sets; however, Latin 1 and ANSI differ in a few character mappings. The ANSI versions of the Win32 Internet functions do not perform any mappings between ANSI and Latin 1—they return what they receive from the server, and pass to the server exactly what they receive from the application. The Win32 Internet functions provide both ANSI and Unicode entry points to facilitate Unicode-only applications.
- Provides full access to Internet protocols.

  Occasionally, applications will need to access extended features of the Internet protocols. The Win32 Internet functions help provide this access.
- Enables high-performance, multithreaded Internet applications.

  The Win32 Internet functions are fully "re-entrant" and multithread safe. Multithreaded applications can make simultaneous calls into the functions from different threads without adverse effects. The

Internet functions themselves complete any necessary synchronization.

- Has persistent caching support built in.

  The Win32 Internet functions provide persistent caching for all of the protocols so the application developer concentrate on obtaining the data and not worrying about managing the cache. For more information about how WinInet uses the cache functions to get proper web behavior, see Persistent URL Cache Functions.

The Win32 Internet functions are intended to make Internet client applications easier to write; they are not intended to facilitate writing Internet servers. This is because servers must be able to control how the protocol is accessed and how I/O is performed in order to achieve the high performance necessary for high-traffic servers. In addition, the Win32 Internet functions are not intended to solve the general issue of access to mail and news servers.

Overview of the Win32 Internet Functions

The following table summarizes the Win32 Internet functions. It uses indenting to show the dependencies among functions. A dependent function can be called only after the related higher-level function is called. This is because the higher-level function returns a handle and sets up a state at the protocol level which is a prerequisite to the successful execution of the dependent function or functions.

InternetOpen	Initializes the application's use of the Win32 Internet functions.
InternetOpenUrl	Begins retrieving an FTP, Gopher, or HTTP URL.
InternetReadFile	Reads URL data.
InternetSetFilePointer	Sets the position for the next read in a file.
InternetCloseHandle	Stops reading data from the URL.
InternetSetStatusCallback	Sets a callback function that is called with status information.
InternetQueryOption	Queries the setting of an Internet option.
InternetSetOption	Sets an Internet option.
InternetCrackUrl	Parses a URL string into components.
InternetCreateUrl	Creates a URL string from components.
InternetCanonicalizeUrl	Converts a URL to a canonical form.
InternetCombineUrl	Combines base and relative URLs.
InternetErrorDlg	Displays predefined dialog boxes for common Internet error conditions.
InternetConfirmZoneCrossing	Checks for changes between secure and non-secure URLs.
InternetTimeFromSystemTime	Formats a date and time according to the specified RFC format.
InternetAttemptConnect	Allows an applicaton to attempt to connect to the Internet before issuing any requests.

FTP

InternetConnect	Opens an FTP session with a server and log on the user.
FtpFindFirstFile	Starts file enumeration in the current directory.
InternetFindNextFile	Continues file enumeration.
InternetCloseHandle	Ends directory file enumeration.
FtpGetFile	Retrieves an entire file from the server.
FtpPutFile	Writes an entire file to the server.
FtpDeleteFile	Deletes a file on the server.
FtpRenameFile	Renames a file on the server.
FtpOpenFile	Initiates access to a file on the server for either reading or writing.
InternetQueryDataAvailable	Queries the amount of data available
InternetReadFile	Reads data from an open file.
InternetWriteFile	Writes data to an open file.
InternetCloseHandle	Ends a read or write operation to or from a file on the server.

FtpCreateDirectory	Creates a new directory on the server.
FtpRemoveDirectory	Deletes a directory on the server.
FtpSetCurrentDirectory	Changes the client's current directory on the server.
FtpGetCurrentDirectory	Returns the client's current directory on the server.
InternetGetLastResponseInfo	Retrieves the text of the server's response to the FTP command.
InternetCloseHandle	Closes the FTP session.

Gopher

InternetConnect	Indicates the Gopher server that the application is interested in accessing.
GopherFindFirstFile	Starts enumerating a Gopher directory listing.
InternetFindNextFile	Continues enumerating a Gopher directory listing.
InternetCloseHandle	Stops enumerating a Gopher directory listing.
GopherOpenFile	Starts retrieving a Gopher object.
InternetQueryDataAvailable	Queries the amount of data available.
InternetReadFile	Reads data from a Gopher object.
InternetCloseHandle	Completes the reading of a Gopher object.
GopherCreateLocator	Forms a Gopher locator for use in other Gopher function calls.
GopherGetAttribute	Retrieves attribute information on the Gopher object.
InternetCloseHandle	Indicates that the application is no longer interested in the server.

HTTP (World Wide Web)

InternetConnect	Indicates the HTTP server the application is interested in accessing.
HttpOpenRequest	Opens an HTTP request handle.
HttpAddRequestHeaders	Adds HTTP request headers to the HTTP request handle.
HttpSendRequest	Sends the specified request to the HTTP server.
InternetQueryDataAvailable	Queries the amount of data available
InternetReadFile	Reads a block of data from an outstanding HTTP request.
HttpQueryInfo	Queries information about an HTTP request.
InternetCloseHandle	Closes an open HTTP request handle.
InternetCloseHandle	Indicates that the application is no longer interested in the server.
InternetCloseHandle	Completes the application's use of the Win32 Internet functions.

Handles

The handles returned by the Win 32 Internet functions are not native system handles; they are only intended to control Internet functions. In other words, Win32 routines such as **ReadFile** and **CloseHandle** will not work on Internet handles. Only Internet handles can be used for the corresponding Internet functions.

Any of the handles returned from the Internet functions can be closed with the InternetCloseHandle function. If a handle is closed while there is still activity on the handle, the activity will be aborted. Note that failing to close valid handles returned from the Internet functions can result in resource leaks.

Handle Hierarchy

Handles returned from Win32 Internet functions are maintained in a hierarchical configuration. The first handle opened is through the InternetOpen function. A handles returned from the InternetConnect function is descended from the handle returned from InternetOpen. File and directory handles are descended from InternetConnect handles by functions such as HttpOpenRequest or FtpFindFirstFile.

Error Handling

The Win32 Internet functions return error information in the same way as Win32 functions. Return values tell whether the function is successful or not. For example, some Internet functions return a BOOL value that is **TRUE** if the function succeeded or **FALSE** if it failed, and others return a handle of type **HINTERNET**. A NULL handle indicates that the function failed, and any other value indicates that it succeeded.

If a function fails, the application can call the Win32 Internet **GetLastError** function to retrieve the specific error code for the failure. In addition, the FTP and Gopher protocols let servers return additional error information. For these protocols, applications can use the InternetGetLastResponseInfo function to retrieve error text.

Both **GetLastError** and InternetGetLastResponseInfo operate on a per-thread basis. If two threads call Internet functions at the same time, error information will be returned for each of the individual threads so that there is no conflict between the threads. However, only the thread that successfully made the call will receive the error information for the function.

Multithreaded Access

The Win32 Internet functions are "re-entrant" in the sense that there can be multiple calls to an individual function from different threads. The functions complete any necessary synchronization. However, multiple simultaneous calls using the same Internet connection can lead to unpredictable results.

For example, if an application has used FtpOpenFile to begin downloading a file from an FTP server and two threads simultaneously make calls to InternetReadFile, there is no guarantee which call will be completed first, or which thread will receive file data first. Applications that use multiple threads for the same Internet connection are responsible for synchronization between threads to ensure a predictable return of information.

Canceling Requests

Win32 Internet functions are synchronous. Sometimes, an application can cancel an outstanding request because of user action. A request can be canceled by using the InternetCloseHandle function to close the handle. Canceling a request in this manner aborts the connection to the server and requires the application to re-establish the connection in order to communicate further.

Unicode Support

All Win32 Internet functions using string arguments on input or output have both ANSI and Unicode versions. As with all Win32 functions, the ANSI functions have "A" as the final character of their name while the Unicode functions have "W." In Windows 95 and Windows NT, both the ANSI and Unicode versions of the functions are implemented.

Because the underlying Internet protocols pass all information in Latin 1, the Unicode versions of the Win32 Internet functions must do translations to and from ANSI. In some cases, it is not possible to convert a Unicode string to ANSI, so the Unicode functions can fail. In the cases where the translation fails, there is usually no ANSI equivalent, so the requested object could not exist.

The Win32 Internet functions do not translate sent or received content. For example, when calling a Unicode function to retrieve a file from an FTP server, the application must specify the filename in Unicode. However, the file data is returned to the application exactly as the FTP server has stored it.

API Flags

Many of the Win32 Internet functions accept a double-word array of flags as a parameter. Any of the defined flags can be passed to any function that accepts flags, but typically only a subset are meaningful to a particular function. If a flag is not meaningful to a function, the function does NOT return an error value. Following is a list an brief description of the defined flags:

INTERNET_FLAG_RELOAD

Force a download of the requested file, object, or directory listing from the origin server, not from the cache

INTERNET_FLAG_RAW_DATA

Return the data in non-HTML format. Only the InternetOpenUrl function uses this flag.

INTERNET_FLAG_EXISTING_CONNECT

Attempt to use an existing InternetConnect object if one exists with the same attributes required to make the request. Only with InternetOpenUrl function uses this flag.

INTERNET_FLAG_ASYNC

Make only asynchronous requests on handles descended from the handle returned from this function. Only the InternetOpen function uses this flag.

INTERNET_FLAG_PASSIVE

Use passive FTP semantics. Only the InternetConnect and InternetOpenUrl functions use this flag.

INTERNET_FLAG_NO_CACHE_WRITE

Do not add the returned entity to the cache.

INTERNET_FLAG_MAKE_PERSISTENT

Add the returned entity to the cache as a persistent entity. This means that standard cache cleanup, consistency checking, or garbage collection can not remove this item from the cache.

INTERNET_FLAG_OFFLINE

Do not make network requests. All entities are returned from the cache. If the requested item is not in the cache, a suitable error, such as **ERROR_FILE_NOT_FOUND** is returned. Only the InternetOpen function uses this flag.

INTERNET_FLAG_SECURE

Use secure transaction semantics. This translates to using SSL/PCT and is only meaningful in HTTP requests.

INTERNET_FLAG_KEEP_CONNECTION

Use keep-alive semantics, if available, for the connection. (This flag is currently NOT IMPLEMENTED.)

INTERNET_FLAG_NO_AUTO_REDIRECT

Do not automatically handle redirection in HttpSendRequest. Only HTTP uses this flag.

INTERNET_FLAG_IGNORE_CERT_CN_INVALID

Disable WinInet checking of SSL/PCT based certificates that are returned from the server against the host name given in the request. WinInet uses a simple check against Certificates by comparing for matching host names and simple wildcarding rules. For security reasons, WinInet does not support full regular expression checking like other clients. This may generate errors on some secure sites.

INTERNET_FLAG_IGNORE_CERT_DATE_INVALID

Disable WinInet checking of SSL/PCT based certificates for proper validity dates.

INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTPS

Disable WinInet's ability to detect this special type of redirect. When this flag is used, WinInet transparently allows redirects from HTTP to HTTPS URLs.

INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTP

Disable WinInet's ability to detect this special type of redirect. When this flag is uses, WinInet transparently allows redirects from HTTPS to HTTP URLs.

INTERNET_FLAG_READ_PREFETCH

Download the entity without requiring the application to initiate reads from the network.

INTERNET_FLAG_NO_COOKIES

Do not automatically add cookie headers to requests, and do not automatically add returned cookies to the cookie database.

INTERNET_FLAG_NO_AUTH

Do not attempt authentication automatically.

INTERNET_FLAG_TRANSFER_ASCII

Transfer file as ASCII (FTP only).

INTERNET_FLAG_TRANSFER_BINARY

Transfer file as binary (FTP only).

Context Values

Many of the Win32 Internet functions that create a handle can also accept an application-defined context value. This context value is associated with the handle until it is closed. For example, you can specify a context value to the HttpOpenRequest function that will be used in all callbacks made for requests against this handle. If the INTERNET_FLAG_ASYNC flag is specified, supplying a zero context value forces the request to be synchronous.

Asynchronous Support

By default, the Win32 Internet functions operate synchronously. An application can request asynchronous operation by setting the INTERNET_FLAG_ASYNC flag in the call to the InternetOpen function. All future calls made against handles derived from the handle returned from InternetOpen will be made asynchronously.

The rationale for asynchronous versus synchronous operation is to allow a single-threaded application to maximize its utilization of the CPU without having to wait for network I/O to complete. Therefore, depending on the request, the operation may complete synchronously or asynchronously. The application should check the return code. If a function returns **FALSE** or NULL, and **GetLastError** returns **ERROR_IO_PENDING**, the request has been made asynchronously and the application will be called back with the INTERNET_STATUS_REQUEST_COMPLETE when the function has completed.

For an application to be able to make requests asynchronously, it must have set the INTERNET_FLAG_ASYNC flag in the call to InternetOpen, it must have registered a valid callback function, and it must supply a non-zero context value.

Persistent Caching

WinInet has built-in caching support that is simple yet flexible. Any data that comes down the wire is cached on the hard disk and retrieved for subsequent requests. The caller has the option of controlling the caching on a per-request basis. In the case of HTTP, most headers coming down the wire are also cached. When an HTTP request is satisfied from the cache, the cached headers are also returned to the caller. This makes data download from WinInet appear seamless, whether it is coming from the cache or from the wire.

General Win32 Internet Functions

The general Win32 Internet functions perform basic Internet file manipulations.

InternetAttemptConnect

Attempts to make a connection to the Internet. This function allows an application to first attempt to connect before issuing any requests. If the connection fails, the application can enter off-line mode.

DWORD InternetAttemptConnect(
  DWORD *dwReserved* **);**

);

Parameters

dwReserved
  Reserved; must be zero.

Return Values

Returns **ERROR_SUCCESS** is successful, or a Win32 error value otherwise.

InternetCanonicalizeUrl

Converts a URL to a canonical form.

BOOL InternetCanonicalizeUrl(
IN LPCTSTR *lpszUrl*,
OUT LPTSTR *lpszBuffer*,
IN OUT LPDWORD *lpdwBufferLength*,
IN DWORD *dwFlags* **);**

Parameters

lpszUrl
Address of the input URL to canonicalize.

lpszBuffer
Address of the buffer that receives the resulting canonicalized URL.

lpdwBufferLength
Length, in bytes, of the *lpszBuffer* buffer. If the function succeeds, this parameter receives the length of the *lpszBuffer* buffer—the length does not include the terminating null. If the function fails, this parameter receives the required length, in bytes, of the *lpszBuffer* buffer—the required length includes the terminating null.

dwFlags
Flags controlling canonicalization. Can be one of these values:

Value	Meaning
ICU_DECODE	Convert %XX escape sequences to characters.
ICU_NO_ENCODE	Don't convert unsafe characters to escape sequence.
ICU_NO_META	Don't convert .. etc. meta path sequences.
ICU_ENCODE_SPACES_ONLY	Encode spaces only.
ICU_BROWSER_MODE	Don't encode characaters after #' or '?', and don't remove trailing whitespace after '?'. If this value is not specified, the entire URL is encoded and trailing whitespace is removed.

If no flags are specified, the function converts all unsafe characters to escape sequences, and to handle meta sequences (\.,\ .., \..., etc).

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible errors include:

Value Meaning

ERROR_INVALID_PARAMETER
Bad string, buffer, buffer size, or flags parameter.

ERROR_INSUFFICIENT_BUFFER
The **lpdwBufferLength* parameter is the size, in bytes, of the buffer required to hold the resultant, canonicalized URL.

ERROR_BAD_PATHNAME
The URL could not be canonicalized.

ERROR_INTERNET_INVALID_URL

The format of the URL is invalid.

Remarks

This function will handle arbitrary protocol schemes, but in order to do so, must infer the unsafe character set.

InternetCloseHandle

Closes any Internet handle that an application has opened.

BOOL InternetCloseHandle(
  IN HINTERNET *hInet* **);**

Parameters

hInet
  Valid Internet handle to be closed.

Return Values

Returns **TRUE** if the handle is successfully closed, or **FALSE** otherwise. To get extended error information, call **GetLastError**.

Remarks

Use this function to close any Internet handle of the type **HINTERNET** and free any associated resources. The function terminates any pending operations on the handle and discards any outstanding data. If a thread is blocking a call to the Internet DLL, another thread in the application can call **InternetCloseHandle** on the Internet handle being used by the first thread to cancel the operation and unblock the first thread.

When an application has finished using the Internet DLL, it should use **InternetCloseHandle** to close the handle returned from InternetOpen.

If asynchronous operations are pending on the handle when it is closed, those operations may complete asynchronously, with a success or failure status. An application should be prepared to accept such indications for canceled requests after the handle has been closed.

If a call to **InternetCloseHandle** is successful, but the handle is still active because of pending asynchronous requests, the handle is invalidated and no further operations can be initiated against that handle. An application can call **GetLastError** to determine if requests are pending. If **GetLastError** returns **ERROR_IO_PENDING**, there were outstanding requests when the handle was closed.

If the application has registered a callback function, the system passes the INTERNET_STATUS_HANDLE_CLOSING value to the callback function for each handle that is closed. Passing this value to the callback function informs the application that the associated handle is being destroyed, and is a final confirmation of handle termination

See Also

FtpFindFirstFile, FtpOpenFile, GopherFindFirstFile, HttpOpenRequest, InternetConnect, InternetOpen

InternetCombineUrl

Combines a base and relative URL into a single URL. The resultant URL will be canonicalized (see InternetCanonicalizeUrl).

BOOL InternetCombineUrl(
IN LPCTSTR *lpszBaseUrl***,**
IN LPCTSTR *lpszRelativeUrl***,**
OUT LPTSTR *lpszBuffer***,**
IN OUT *lpdwBufferLength***,**
IN DWORD *dwFlags* **);**

Parameters

lpszBaseUrl
Address of the base URL to be combined.

lpszRelativeUrl
Address of the relative URL to be combined.

lpszBuffer
Address of a buffer that receives the resulting URL.

lpdwBufferLength
Size, in bytes, of the *lpszBuffer* buffer. If the function succeeds, this parameter receives the length, in characters. of the resultant combined URL—the length does not include the null terminator. If the function fails, this parameter receives the length, in bytes, of the required buffer—the length includes the null terminator.

dwFlags
Flags controlling the operation of the function. For a description of the flags, see InternetCanonicalizeUrl.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error codes include:

Value Meaning

ERROR_INVALID_PARAMETER
Bad string, buffer, buffer size, or flags parameter.

ERROR_INSUFFICIENT_BUFFER
The **lpdwBufferLength* parameter, in bytes, of the buffer required to hold the resultant, combined URL.

ERROR_BAD_PATHNAME
The URLs could not be combined.

ERROR_INTERNET_INVALID_URL
The format of the URL is invalid.

InternetConnect

Opens an FTP, Gopher, or HTTP session for a given site.

HINTERNET InternetConnect(
IN HINTERNET *hInternetSession*,
IN LPCTSTR *lpszServerName*,
IN INTERNET_PORT *nServerPort*,
IN LPCTSTR lpszUsername *OPTIONAL*,
IN LPCTSTR lpszPassword *OPTIONAL*,
IN DWORD *dwService*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hInternetSession
Handle of the current Internet session. The handle must have been returned by a previous call to InternetOpen.

lpszServerName
Address of a null-terminated string that contains the host name of an Internet server. Alternately, the string can contain the IP number of the site in ASCII dotted-decimal format (for example, 11.0.1.45).

nServerPort
Number of the TCP/IP port on the server to connect to. Can be one of the values in the following list. If this parameter is set to **INTERNET_INVALID_PORT_NUMBER**, the function uses the default port for the specified service.

Value	Meaning
INTERNET_DEFAULT_FTP_PORT	Use the default port for FTP servers (port 21).
INTERNET_DEFAULT_GOPHER_PORT	Use the default port for Gopher servers (port 70).
INTERNET_DEFAULT_HTTP_PORT	Use the default port for HTTP servers (port 80).
INTERNET_DEFAULT_HTTPS_PORT	Use the default port for HTTPS servers (port 443).

lpszUsername
Address of a null-terminated string that contains the name of the user to log in. If this parameter is NULL, the function uses an appropriate default. For the FTP protocol, the default is anonymous.

lpszPassword
Address of a null-terminated string that contains the password to use to log in. If both *lpszPassword* and *lpszUsername* are NULL, the function uses the default anonymous password. In the case of FTP, the default anonymous password is the user's email name. If *lpszPassword* is NULL (or an empty string), but *lpszUsername* is not NULL, the function uses a blank password. The following table describes the behavior for the four possible settings of *lpszUsername* and *lpszPassword*:

lpszUsername	*lpszPassword*	User name sent to FTP server	Password sent to FTP server
NULL or ""	NULL or ""	"anonymous"	User's email name
Non-NULL string	NULL or ""	*lpszUsername*	""
NULL	Non-NULL	ERROR	ERROR

	string		
Non-NULL string	Non-NULL string	*lpszUsernam e*	*lpszPassword*

dwService

Type of service to access. Can be one of these values:

Value	Meaning
INTERNET_SERVICE_FTP	FTP service
INTERNET_SERVICE_GOPHER	Gopher service
INTERNET_SERVICE_HTTP	HTTP service

dwFlags

Flags specific to the service used. Can be one of these values:

If *dwService* is:	*dwFlags* supported
INTERNET_SERVICE_FTP	**INTERNET_CONNECT_FLAG_PASSIVE** Use passive mode in all data connections for this FTP session.

dwContext

An application-defined value that is used to identify the application context for the returned handle in callbacks.

Return Values

Returns a valid handle to the FTP, Gopher, or HTTP session if the connection is successful, or NULL otherwise. To get extended error information, call **GetLastError**. An application can also use InternetGetLastResponseInfo to determine why access to the service was denied.

Remarks

The **InternetConnect** function is required before communicating with any Internet service.

Having a connect function for all protocols, even those that do not use persistent connections, lets an application communicate common information about several requests using a single function call. In addition, this allows for future versions of Internet protocols that do not require a connection to be established for every client request.

For some protocols, **InternetConnect** actually establishes a connection with the server while for others, such as Gopher, the actual connection is not established until the application requests a specific transaction.

For maximum efficiency, applications using the Gopher and HTTP protocols should try to minimize calls to **InternetConnect** and avoid calling this function for every transaction requested by the user. One way to accomplish this is to keep a small cache of handles returned from **InternetConnect**, and when the user makes a request to a previously accessed server, that session handle is still available.

An application that needs to display multiline text information sent by an FTP server can use InternetGetLastResponseInfo to retrieve the text.

For FTP connections, if *lpszUsername* is NULL, **InternetConnect** sends the string "anonymous" as the user name. If *lpszPassword* is NULL, **InternetConnect** attempts to use the user's email name as the password.

To close the handle returned from **InternetConnect**, the application should call InternetCloseHandle. This function disconnects the client from the server and free all resources associated with the connection.

See Also

InternetCloseHandle, InternetOpen

InternetConfirmZoneCrossing

Checks for changes between secure and non-secure URLs. When a change occurs in security between two URLs, an application should allow the user to acknowledge this change, typically by displaying a dialog box.

BOOL InternetConfirmZoneCrossing(
IN HWND *hWnd*,
IN LPSTR *szUrlPrev*,
IN LPSTR *szUrlNew*,
IN BOOL *bPost* **);**

Parameters

hWnd
Handle of the parent window for any needed dialog box.

szUrlPrev
URL that was viewed before the current request was made.

szUrlNew
URL that the user is about to request and view.

bPost
TRUE if a post is being made in this request.

InternetCrackUrl

Cracks a URL into its component parts.

BOOL InternetCrackUrl(
IN LPCSTR *lpszUrl*,
IN DWORD *dwUrlLength*,
IN DWORD *dwFlags*,
IN OUT LPURL_COMPONENTS *lpUrlComponents* **);**

Parameters

lpszUrl
Address of a string that contains the URL to crack.

dwUrlLength
Length of the *lpszUrl* string, or zero if *lpszUrl* is an ASCIIZ string.

dwFlags
Flags controlling the operation. Can be one of these values:

Value	Meaning
ICU_ESCAPE	Convert unsafe characters in the URL-path component to escape sequences.
ICU_USERNAME	When adding the user name, use the name that was specified at login time.

lpUrlComponents
Address of a URL_COMPONENTS structure that receives the URL components:

Return Values

Returns **TRUE** if the function succeeds, or **FALSE** otherwise. To get extended error information, call **GetLastError**. If the function can find no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

Remarks

The required components are indicated by the members of the URL_COMPONENTS structure: if both the input pointer and length members are zero for a given component, that component is not returned. If the input pointer member is NULL, but the length member is non-zero, on output the pointer member contains the address of the corresponding component in the *lpszUrl* string, and the length member contains the length of the component. If both the input pointer and length members are not zero, the pointer member contains the address of a user-supplied buffer where the component is copied; the length member is updated with the length of the copied component, minus 1 for the trailing string terminator.

See Also

FtpOpenFile, InternetCloseHandle, InternetFindNextFile, InternetSetStatusCallback

InternetCreateUrl

Creates a URL from its component parts.

BOOL InternetCreateUrl(
IN LPURL_COMPONENTS *lpUrlComponents***,**
IN DWORD *dwFlags***,**
OUT LPSTR *lpszUrl***,**
IN OUT LPDWORD *lpdwUrlLength* **);**

Parameters

lpUrlComponents
Address of a URL_COMPONENTS structure that contains the components from which to create the URL.

dwFlags
Flags that control the operation of this function. Can be a combination of these values:

Value	Meaning
ICU_ESCAPE	Convert unsafe characters in the URL-path component to escape sequences.
ICU_USERNAME	When adding the user name, use the name that was specified at login time.

lpszUrl
Address of a buffer that receives the URL.

lpdwUrlLength
Length, in bytes, of the *lpszUrl* buffer. When the function returns, this parameter receives the length, in bytes, of the URL string, minus one for the terminating character. If **GetLastError** returns **ERROR_INSUFFICIENT_BUFFER**, this parameter receives the number of bytes required to hold the created URL.

Return Values

Returns **TRUE** if the function succeeds, or **FALSE** otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

InternetErrorDlg

Displays a dialog box that explains why an error occurred. An application can call this function for several different error codes, and can do bookkeeping based on the user's response to the dialog box.

DWORD InternetErrorDlg(
IN HWND *hWnd*,
IN OUT HINTERNET *hInternet*,
IN DWORD *dwError*,
IN DWORD *dwFlags*,
IN OUT LPVOID **lppvData* **);**

Parameters

hWnd
Handle of the parent window for any needed dialog box. This parameter can be NULL if no dialog box is needed.

hInternet
Handle of the Internet connection used in the call to HttpSendRequest.

dwError
Error value for which to display a dialog box. Can be one of these values:

Value **Meaning**

ERROR_INTERNET_HTTP_TO_HTTPS_ON_REDIR
notifies the user of the zone crossing to and from a secure site.

ERROR_INTERNET_INCORRECT_PASSWORD
Displays a dialog box for obtaining the user's name and password. (On Windows 95, the function attempts to use the network caching user interface and disk cache.)

ERROR_INTERNET_SEC_CERT_CN_INVALID
Displays an Invalid SSL Common Name dialog box, and lets the user view the incorrect certificate.

ERROR_INTERNET_POST_IS_NON_SECURE
Displays a warning about posting data to the server through a non-secure connection.

ERROR_INTERNET_SEC_CERT_DATE_INVALID
Tells the user that the SSL certificate has expired.

dwFlags
Action flags. Can be a combination of these values:

Value **Meaning**

FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS
If the function succeeds, store the results of the dialog box in the Internet handle.

FLAGS_ERROR_UI_FLAGS_GENERATE_DATA
Query the Internet handle for needed information. The function constructs the appropriate data structure for the error. (For example, for Cert CN failures, the function will grab the certificate.)

FLAGS_ERROR_UI_FLAGS_NO_UI
Perform any default behavior needed to restart the request (initiated by calling HttpSendRequest) without generating a Windows-based dialog box.

FLAGS_ERROR_UI_FLAGS_FILTER_FOR_ERRORS

Scan for errors. Call after every SendRequest. The function detects any relevant errors and generates a dialog box or returns if no action is needed.

lppvData

Address of a pointer to a yet-to-be-defined structure. The structure may be different for each error that needs to be handled

Return Values

Returns **ERROR_SUCCESS**, **ERROR_CANCELLED**, or ERROR_INTERNET_FORCE_RETRY.

InternetFindNextFile

Continues a file search started as a result of a previous call to FtpFindFirstFile or GopherFindFirstFile.

BOOL InternetFindNextFile(
IN HINTERNET *hFind*,
OUT LPVOID *lpvFindData* **);**

Parameters

hFind
Valid handle returned from either FtpFindFirstFile or GopherFindFirstFile.

lpvFindData
Address of a buffer that receives information about the found file or directory. The format of the information placed in the buffer depends on the protocol in use. The FTP protocol returns a **WIN32_FIND_DATA** structure and the Gopher protocol returns a GOPHER_FIND_DATA structure.

Return Values

Returns **TRUE** if the function succeeds, or **FALSE** otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

See Also

FtpFindFirstFile, GopherFindFirstFile

InternetGetLastResponseInfo

Retrieves error text from the last Win32 Internet function that failed.

BOOL InternetGetLastResponseInfo(
OUT LPDWORD *lpdwError*,
OUT LPTSTR lpszBuffer *OPTIONAL*,
IN OUT LPDWORD *lpdwBufferLength* **);**

Parameters

lpdwError
Address of a variable that receives an error code pertaining to the operation that failed.

lpszBuffer
Address of a buffer that receives the error text.

lpdwBufferLength
Size of the *lpszBuffer* buffer. When the function returns, this parameter contains the size of the string written to the buffer.

Return Values

Returns **TRUE** if error text was successfully written to the buffer, or **FALSE** otherwise. To get extended error information, call **GetLastError**. If the buffer is too small to hold all the error text, **GetLastError** returns ERROR_INSUFFICIENT_BUFFER, and the *lpdwBufferLength* parameter contains the minimum buffer size required to return all the error text.

Remarks

The FTP and Gopher protocols can return additional text information along with most errors. This extended error information can be retrieved by using the **InternetGetLastResponseInfo** function whenever a **GetLastError** returns ERROR_INTERNET_EXTENDED_ERROR (occurring after an unsuccessful function call).

The buffer pointed to by *lpszBuffer* must be large enough to hold both the error string and a zero terminator at the end of the string. However, note that the value returned in *lpdwBufferLength* does not include the terminating zero.

InternetOpen

Initializes an application's use of the Win32 Internet functions.

HINTERNET InternetOpen(
IN LPCTSTR *lpszAgent*,
IN DWORD *dwAccessType*,
IN LPCTSTR lpszProxyName *OPTIONAL*,
IN LPCSTR lpszProxyBypass *OPTIONAL*,
IN DWORD *dwFlags* **);**

Parameters

lpszAgent
Address of a string that contains the name of the application or entity calling the Internet functions (for example, Microsoft Internet Explorer). This name is used as the user agent in the HTTP protocol.

dwAccessType
Type of access required. Can be one of these values:

Value	Meaning
INTERNET_OPEN_TYPE_DIRECT	Resolve all host names locally.
INTERNET_OPEN_TYPE_PROXY	Pass requests to the proxy unless a proxy bypass list is supplied and the name to be resolved bypasses the proxy. In this case, the function proceeds as for **INTERNET_OPEN_TYPE_DIRECT**.
INTERNET_OPEN_TYPE_PRECONFIG	The proxy or direct configuration is retrieved from the registry.

lpszProxyName
Address of a string that contains the name of the proxy server (or servers) to use if proxy access was specified. If this parameter is NULL, the function reads proxy information from the registry. For more information about this parameter, see the comments below.

lpszProxyBypass
Address of an optional list of host names or IP addresses, or both, that are known locally. Requests to these names are not routed through the proxy. The list can contain wildcards, such as "157.55.* *int*", meaning any IP address starting with 157.55, or any name containing the substring "int", will bypass the proxy.
If this parameter specifies the "<local>" macro as the only entry, the function bypasses any host name that does not contain a period. For example, "www.microsoft.com" would be routed to the proxy, whereas "internet" would not. p>If this parameter is NULL, the function reads the bypass list from the registry.

dwFlags
Flag that indicates various options affecting the behavior of the function. Can be a combination of these values:

Value	Meaning
INTERNET_FLAG_OFFLINE	Satisfy download operations on this handle through the persistent cache only. If the item does not exist in the cache, the function returns an appropriate error code.
INTERNET_FLAG_ASYNC	Future operations on this handle may fail with

ERROR_IO_PENDING. A status callback will be made with INTERNET_STATUS_REQUEST_COMPLETE. This callback will be on a thread other than the one for the original request. A status callback routine must be registered or the functions will be completed synchronously.

Return Values

Returns a valid handle that the application passes on to subsequent Win32 Internet functions. If **InternetOpen** fails, it returns NULL. To get a specific error code, call **GetLastError**.

Remarks

This function is the first Win32 Internet function called by an application. It tells the Internet DLL to initialize internal data structures and prepare for future calls from the application. When the application finishes using the Internet functions, it should call InternetCloseHandle to free the handle and any associated resources.

If *dwFlags* includes **INTERNET_FLAG_ASYNC**, all handles derived from this handle will have asynchronous behavior as long as a status callback routine is registered. For a function to be completed synchronously, the *dwContext* must be set to zero for that call.

By default, the function assumes that the proxy specified by *lpszProxyName* is a CERN proxy. For example, "proxy" defaults to a CERN proxy called "proxy" that listens at port 80 (decimal). An application can specify more than one proxy, including different proxies for the different protocols. For example, if you specify "ftp=ftp://ftp-gw gopher=http://jericho:99 proxy", FTP requests are made through the FTP proxy "ftp-gw" which listens at port 21 (default for FTP), and gopher requests are made through a CERN proxy called "jericho" which listens at port 99. All other requests (for example, HTTP requests) are made through the CERN proxy called "proxy" which listens at port 80. Note that if the application is only using FTP, for example, it would not need to specify "ftp=ftp://ftp-gw:21". It could specify just "ftp-gw". An application must specify the protocol names only if it will be using more than one protocol per handle returned by **InternetOpen**.

The application can make any number of calls to **InternetOpen**, although a single call is normally sufficient. The application may need to have separate behaviors defined for each **InternetOpen** instance, such as different proxy servers configured for each.

See Also

InternetCloseHandle

InternetOpenUrl

The **InternetOpenUrl** function begins reading an FTP, Gopher, or HTTP Universal Resource Locator (URL).

HINTERNET InternetOpenUrl(
IN HINTERNET *hInternetSession*,
IN LPCTSTR *lpszUrl*,
IN LPCTSTR lpszHeaders *OPTIONAL*,
IN DWORD *dwHeadersLength*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hInternetSession
Handle of the current Internet session. The handle must have been returned by a previous call to InternetOpen.

lpszUrl
Address of a string that contains the URL to begin reading. Only URLs beginning with ftp:, gopher:, http, or https: are supported.

*lpszHeader*s
Address of a string that contains the headers to be sent to the HTTP server. (For more information, see the description of the *lpszHeaders* parameter to HttpSendRequest.)

dwHeadersLength
Length, in characters, of the additional headers. If this parameter is -1L and *lpszHeaders* is not NULL, *lpszHeaders* is assumed to be zero-terminated (ASCIIZ) and the length is calculated.

dwFlags
Action flags. Can be one of these values:

Value	Meaning
INTERNET_FLAG_RELOAD	Get the data from the wire even if it is locally cached.
INTERNET_FLAG_DONT_CACHE	Do not cache the data, either locally or in any gateways.
INTERNET_FLAG_RAW_DATA	Return raw data (**WIN32_FIND_DATA** structures for FTP, and GOPHER_FIND_DATA structures for Gopher). If this flag is not specified, **InternetOpenUrl** returns HTML formatted for directories.
INTERNET_FLAG_SECURE	Request secure transactions on the wire with Secure Sockets Layer or PCT. This flag applies to HTTP requests only.
INTERNET_FLAG_EXISTING_CONNECT	If possible, reuse the existing connections to the server for new requests generated by **InternetOpenUrl** instead of creating a new session for each request.

dwContext
An application-defined value that is passed, along with the returned handle, to any callback functions.

Return Values

Returns a valid handle to the FTP, Gopher, or HTTP URL if the connection is successfully established, or NULL if the connection fails. To get a specific error code, call **GetLastError**. To determine why

access to the service was denied, call InternetGetLastResponseInfo.

Remarks

This is a general function that an application can use to retrieve data over any of the protocols that theWin32 Internet functions support. This function is particularly useful when the application does not need to access the particulars of a protocol, but only requires the data corresponding to a URL. The **InternetOpenUrl** function parses the URL string, establishes a connection to the server, and prepares to download the data identified by the URL. The application can then use InternetReadFile to retrieve the URL data. It is not necessary to call InternetConnect before **InternetOpenUrl**.

Use InternetCloseHandle to close the handle returned from **InternetOpenUrl**. However, note that closing the handle before all the URL data has been read results in the connection being terminated.

See Also

HttpSendRequest, InternetCloseHandle, InternetOpen, InternetReadFile

InternetQueryDataAvailable

Queries the amount of data available.

BOOL InternetQueryDataAvailable(
IN HINTERNET *hFile*,
OUT LPDWORD *lpdwNumberOfBytesAvailable*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hFile
Valid Internet file handle, as returned by FtpOpenFile, GopherOpenFile or HttpOpenRequest.

lpdwNumberOfBytesAvailable
Address of a variable that receives the number of available bytes.

dwFlags
Reserved; must be zero.

dwContext
Reserved; must be zero.

Return Values

Returns **TRUE** if the function succeeds, or **FALSE** otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, the **GetLastError** function returns ERROR_NO_MORE_FILES.

Remarks

This function returns the number of bytes of data that are available to be read immediately by a subsequent call to InternetReadFile. If there is currently no data available and the end of the file has not been reached, the request pends until data becomes available

See Also

FtpFindFirstFile, GopherFindFirstFile

InternetQueryOption

Queries an Internet option on the specified handle.

BOOL InternetQueryOption(
IN HINTERNET hInternet *OPTIONAL*,
IN DWORD *dwOption*,
OUT LPVOID lpBuffer *OPTIONAL*,
IN OUT LPDWORD *lpdwBufferLength* **);**

Parameters

hInternet
Internet handle on which to query information.

dwOption
Internet option to query. Can be one of these values:

Value Meaning

INTERNET_OPTION_CALLBACK
Returns the address of the callback function defined for this handle.

INTERNET_OPTION_CONNECT_TIMEOUT
The time-out value in milliseconds to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_CONNECT_RETRIES
The retry count to use for Internet connection requests. If a connection attempt still fails after the specified number of tries, the request is canceled. The default is five retries.

INTERNET_OPTION_CONNECT_BACKOFF
The delay value in milliseconds to wait between connection retries. (This flag is currently NOT IMPLEMENTED.)

INTERNET_OPTION_CONTROL_SEND_TIMEOUT
The time-out value in milliseconds to use for non-data (control) Internet send requests. If a non-data send request takes longer than this time-out, the request is canceled. The default time-out is infinite. Currently, this value only has meaning for FTP sessions.

INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT
The time-out value in milliseconds to use for non-data (control) Internet receive requests. If a non-data receive request takes longer than this time-out value, the request is canceled. The default time-out is infinite. Currently, this value only has meaning for FTP sessions.

INTERNET_OPTION_DATA_SEND_TIMEOUT
The time-out value in milliseconds to use for Internet data send requests. If a data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_DATA_RECEIVE_TIMEOUT
The time-out value in milliseconds to use for data Internet receive requests. If a data receive request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_HANDLE_TYPE
Returns the type of the Internet handle passed in. Possible return values include:

- **INTERNET_HANDLE_TYPE_INTERNET**
- **INTERNET_HANDLE_TYPE_CONNECT_FTP**
- **INTERNET_HANDLE_TYPE_CONNECT_GOPHER**
- **INTERNET_HANDLE_TYPE_CONNECT_HTTP**
- **INTERNET_HANDLE_TYPE_FTP_FIND**
- **INTERNET_HANDLE_TYPE_FTP_FIND_HTML**
- **INTERNET_HANDLE_TYPE_FTP_FILE**
- **INTERNET_HANDLE_TYPE_FTP_FILE_HTML**
- **INTERNET_HANDLE_TYPE_GOPHER_FIND**
- **INTERNET_HANDLE_TYPE_GOPHER_FIND_HTML**
- **INTERNET_HANDLE_TYPE_GOPHER_FILE**
- **INTERNET_HANDLE_TYPE_GOPHER_FILE_HTML**
- **INTERNET_HANDLE_TYPE_HTTP_REQUEST**

INTERNET_OPTION_CONTEXT_VALUE

Returns the context value associated with this Internet handle.

INTERNET_OPTION_READ_BUFFER_SIZE

Returns the size of the read buffer (for example, the buffer used by FtpGetFile).

INTERNET_OPTION_WRITE_BUFFER_SIZE

Returns the size of the write buffer (for example, the buffer used by FtpPutFile).

INTERNET_OPTION_ASYNC_ID

Returns the identifier of the last asynchronous request made in this thread context.

INTERNET_OPTION_ASYNC_PRIORITY

Returns the priority of this download if it is an asynchronous download.

INTERNET_OPTION_PARENT_HANDLE

Returns the parent handle of this handle.

INTERNET_OPTION_KEEP_CONNECTION

Returns an indication whether this handle uses persistent connections. Can be one of these values:

Value	Meaning
INTERNET_KEEP_ALIVE_UNKNOWN	
INTERNET_KEEP_ALIVE_ENABLED	
INTERNET_KEEP_ALIVE_DISABLED	

INTERNET_OPTION_USERNAME

Returns the user name associated with a handle returned by InternetConnect.

INTERNET_OPTION_PASSWORD

Returns the password associated with the handle returned by InternetConnect.

INTERNET_OPTION_REQUEST_FLAGS

Returns special status flags about the current download in progress. The only flag that is returned at this time is **INTERNET_REQFLAG_FROM_CACHE**. This is the way for the caller to find out whether a request is being satisfied from the cache.

INTERNET_OPTION_EXTENDED_ERROR

Returns the Windows Sockets error code that was mapped to the ERROR_INTERNET_ error codes last returned in this thread context.

INTERNET_OPTION_SECURITY_CERTIFICATE_STRUCT

Returns the certificate for an SSL/PCT server into the INTERNET_CERTIFICATE_INFO structure.

INTERNET_OPTION_SECURITY_CERTIFICATE

Returns the certificate for an SSL/PCT server into a formatted string.

INTERNET_OPTION_SECURITY_KEY_BITNESS

Returns the bit size of the encryption key. The larger the number the greater the encryption strength being used.

INTERNET_OPTION_OFFLINE_MODE

INTERNET_OPTION_CACHE_STREAM_HANDLE

INTERNET_OPTION_ASYNC

INTERNET_OPTION_SECURITY_FLAGS

Returns the security flags for a handle. Can be a combination of these values:

- **SECURITY_FLAG_128BIT**
- **SECURITY_FLAG_40BIT**
- **SECURITY_FLAG_56BIT**
- **SECURITY_FLAG_IETFSSL4**
- **SECURITY_FLAG_IGNORE_CERT_CN_INVALID**
- **SECURITY_FLAG_IGNORE_CERT_DATE_INVALID**
- **SECURITY_FLAG_IGNORE_REDIRECT_TO_HTTP**
- **SECURITY_FLAG_IGNORE_REDIRECT_TO_HTTPS**
- **SECURITY_FLAG_NORMALBITNESS**
- **SECURITY_FLAG_PCT**
- **SECURITY_FLAG_PCT4**
- **SECURITY_FLAG_SECURE**
- **SECURITY_FLAG_SSL**
- **SECURITY_FLAG_SSL3**
- **SECURITY_FLAG_UNKNOWNBIT**

INTERNET_OPTION_DATAFILE_NAME

Returns the name of the file backing a downloaded entity.

INTERNET_OPTION_URL

Returns the full URL of a downloaded entity.

INTERNET_OPTION_REFRESH

Allows variables to be re-read from the registry for a handle.

INTERNET_OPTION_PROXY

Sets or queries the proxy information on an existing InternetOpen handle. The *lpBuffer* parameter is an INTERNET_PROXY_INFO structure that contains the proxy information.

INTERNET_OPTION_VERSION

Returns the version number of WININET.DLL. The *lpBuffer* parameter is the address of an INTERNET_VERSION_INFO structure.

lpBuffer
Address of a buffer that receives the option setting.

lpdwBufferLength
Address of a variable that contains the length of *lpBuffer*. When the function returns, this parameter receives the length of the data placed into *lpBuffer*. If **GetLastError** returns **ERROR_INSUFFICIENT_BUFFER**, this parameter receives the number of bytes required to hold the created URL.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

Use InternetSetOption to select and set the specific option to query.

See Also

FtpGetFile, FtpPutFile, InternetConnect, InternetOpen, InternetSetOption

InternetReadFile

Reads data from a handle opened by the InternetOpenUrl, FtpOpenFile, GopherOpenFile, or HttpOpenRequest functions.

BOOL InternetReadFile(
IN HINTERNET *hFile*,
IN LPVOID *lpBuffer*,
IN DWORD *dwNumberOfBytesToRead*,
OUT LPDWORD *lpNumberOfBytesRead* **);**

Parameters

hFile
Valid handle returned from a previous call to InternetOpenUrl, FtpOpenFile, GopherOpenFile, or HttpOpenRequest.

lpBuffer
Address of a buffer that receives the data read.

dwNumberOfBytesToRead
Number of bytes to read.

lpNumberOfBytesRead
Address of a variable that receives the number of bytes read. The **InternetReadFile** function sets this value to zero before doing any work or error checking.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. An application can also use InternetGetLastResponseInfo when necessary.

Remarks

If the return value is **TRUE** and the number of bytes read is zero, the transfer has been completed and there are no more bytes to read on the handle. This is analogous to reaching EOF in a local file. The application should then call InternetCloseHandle.

The buffer pointed to by *lpBuffer* is not always filled by calls to **InternetReadFile** (sufficient data may not have arrived from the server). When reading HTML data, for the first read posted, the buffer must be large enough to hold the HTML headers. When reading HTML-encoded directory entries, the buffer must be large enough to hold at least one complete entry.

When the item being read is also being cached by WinInet, the application must ensure that a read is made for end-of-file in order that the cache file is committed correctly

This function always fulfills the user's request. If more data is requested than is available, the function waits until enough data to complete the request is available.

The only time that less data is returned than requested is when the end of the file has been reached.

This function can also be used to retrieve FTP and gopher directory listings as a HTML document on a handle opened by InternetOpenUrl. The INTERNET_FLAG_RAW_DATA value must not have been specified in the call to InternetOpenUrl.

See Also

FtpOpenFile, GopherOpenFile, HttpOpenRequest, InternetCloseHandle,

InternetOpenUrl

InternetSetFilePointer

Sets a file position for InternetReadFile. This is a synchronous call; however, subsequent calls to InternetReadFile may block or return pending if the data is not available from the cache and the server does not support random access.

BOOL InternetSetFilePointer(
IN HINTERNET *hFile*,
IN LONG *lDistanceToMover*,
IN PVOID *pReserved*,
IN DWORD *dwMoveMethod*,
IN DWORD *dwContext* **);**

Parameters

hFile
Valid handle returned from a previous call to InternetOpenUrl on an HTTP URL, or to HttpOpenRequest using the GET method and passed to HttpSendRequest. The INTERNET_FLAG_DONT_CACHE value must not have been specified when the handle was created.

lDistanceToMove
Number of bytes to move the file pointer. A positive value moves the pointer forward in the file and a negative value moves it backward.

pReserved
Reserved, must be NULL.

dwMoveMethod
Starting point for the file pointer move. Can be one of these values:

Value	Meaning
FILE_BEGIN	The starting point is zero or the beginning of the file. If FILE_BEGIN is specified, *lDistanceToMove* is interpreted as an unsigned location for the new file pointer.
FILE_CURRENT	The current value of the file pointer is the starting point.
FILE_END	The current end-of-file position is the starting point. This method fails if the content length is unknown.

dwContext
Reserved, must be zero.

Return Values

Returns the current file position if the function succeeds, or -1 otherwise.

See Also

HttpOpenRequest, HttpSendRequest, InternetOpenUrl, InternetReadFile

InternetSetOption

Sets an Internet option on the specified handle.

BOOL InternetSetOption(
  IN HINTERNET hInternetSession *OPTIONAL*,
  IN DWORD *dwOption*,
  IN LPVOID *lpBuffer*,
  IN DWORD *dwBufferLength* **);**

Parameters

hInternetSession
  Internet handle on which to set information.

dwOption
  Internet option to set. Can be a combination of these values:

  Value Meaning

  INTERNET_OPTION_CONNECT_TIMEOUT

    Set the time-out value in milliseconds to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

  INTERNET_OPTION_CONNECT_RETRIES

    The retry count to use for Internet connection requests. If a connection attempt still fails after the specified number of tries, the request is canceled. The default is five retries.

  INTERNET_OPTION_CONNECT_BACKOFF

    The delay value in milliseconds to wait between connection retries.

  INTERNET_OPTION_CONTROL_SEND_TIMEOUT

    The time-out value in milliseconds to use for non-data (control) Internet send requests. If a non-data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite. Currently, this value only has meaning for FTP sessions.

  INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT

    The time-out value in milliseconds to use for non-data (control) Internet receive requests. If a non-data receive request takes longer than this time-out value, the request is canceled. The default time-out value is infinite. Currently, this value only has meaning for FTP sessions.

  INTERNET_OPTION_DATA_SEND_TIMEOUT

    The time-out value in milliseconds to use for data Internet send requests. If a data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

  INTERNET_OPTION_DATA_RECEIVE_TIMEOUT

    The time-out value in milliseconds to use for data Internet receive requests. If a data receive-request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

  INTERNET_OPTION_ASYNC PRIORITY

    Sets the priority of this download if the it is an asynchronous download. This option hasn't been implemented.

  INTERNET_OPTION_CONTEXT_VALUE

Sets the context value associated with this Internet handle.

INTERNET_OPTION_USERNAME

Sets the user name associated with a handle returned by InternetConnect.

INTERNET_OPTION_PASSWORD

Sets the password associated with a handle returned by InternetConnect.

INTERNET_OPTION_READ_BUFFER_SIZE

Size, in bytes, of the buffer to use to read the data.

INTERNET_OPTION_WRITE_DATA

Size, in bytes, of the buffer to use while writing out the data.

lpBuffer

Address of a buffer that contains the option setting.

dwBufferLength

Length of the *lpBuffer* buffer.

Value	Meaning
ISO_GLOBAL	If set, updates the WinInet shared information section.
ISO_REGISTRY	If set, updates the registry for the corresponding registry variable.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

See InternetQueryOption for a list of possible options.

InternetSetOptionEx

Sets an Internet option on the specified handle.

BOOL InternetSetOption(
IN HINTERNET hInternet *OPTIONAL*,
IN DWORD *dwOption*,
IN LPVOID *lpBuffer*,
IN DWORD *dwBufferLength*,
IN DWORD *dwFlags* **);**

Parameters

hInternetSession
Internet handle on which to set information.

dwOption
Internet option to set. For a list of possible values, see InternetSetOption.

lpBuffer
Address of a buffer that contains the option setting.

dwBufferLength
Length of the *lpBuffer* buffer.

dwFlags
Action flags. Can be one of these values:

Value	Meaning
ISO_GLOBAL	Modify the option globally.
ISO_REGISTRY	Write the option to the registry (where applicable).

Return Values

Returns **TRUE**if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

InternetSetStatusCallback

Sets up a callback function that Win32 Internet functions can call as progress is made during an operation.

INTERNET_STATUS_CALLBACK InternetSetStatusCallback(
IN HINTERNET *hInternet*,
IN INTERNET_STATUS_CALLBACK *lpfnInternetCallback* **);**

Parameters

hInternet
Handle for which the callback is to be set.

lpfnInternetCallback
Address of the callback function to call when progress is made, or to return NULL to remove the existing callback function. For more information about the callback function, see InternetStatusCallback.

Return Values

Returns the previously defined status callback function if successful, NULL if there was no previously-defined status callback function, or **INTERNET_INVALID_STATUS_CALLBACK** if the callback function is not valid.

Remarks

Both synchronous and asynchronous functions use the callback function to indicate the progress of the request, such as resolving a name, connecting to a server, and so on. Additionally, the callback function is required for an asynchronous operation because the asynchronous function calls the callback function, passing the INTERNET_STATUS_REQUEST_COMPLETE value to the function to indicate that the request has completed.

A callback function can be set on any handle, and is inherited by derived handles. A callback function can be changed using **InternetSetStatusCallback**, providing there are no pending requests that need to use the previous callback value. Note however, that changing the callback function on a handle does not change the callbacks on derived handles, such as that returned by InternetConnect. You must change the callback function at each level.

Many of the Win32 Internet functions perform several operations on the network. Each operation can take time to complete and each can fail.

It is sometimes desirable to display status information during a long-term operation. You can display status information by setting up an Internet status callback function that cannot be removed as long as any callbacks or any asynchronous functions are pending.

After initiating **InternetSetStatusCallback**, it can be accessed from within any Win32 Internet function for monitoring time-intensive network operations.

InternetStatusCallback

This is a placeholder for the application-defined status callback

VOID InternetStatusCallback(
IN HINTERNET *hInternet*,
IN DWORD *dwContext*,
IN DWORD *dwInternetStatus*,
IN LPVOID lpvStatusInformation *OPTIONAL*,
IN DWORD *dwStatusInformationLength* **);**

Parameters

hInternet
Handle for which the callback function is being called.

dwContext
Application-defined context value associated with *hInternet*

dwInternetStatus
Status code that indicates why the callback function is being called. Can be one of these values:

INTERNET_STATUS_RESOLVING_NAME
Looking up the IP address of the name contained in *lpvStatusInformation*.

INTERNET_STATUS_NAME_RESOLVED
Successfully found the IP address of the name contained in *lpvStatusInformation*.

INTERNET_STATUS_CONNECTING_TO_SERVER
Connecting to the socket address (SOCKADDR) pointed to by *lpvStatusInformation*.

INTERNET_STATUS_CONNECTED_TO_SERVER
Successfully connected to the socket address (SOCKADDR) pointed to by *lpvStatusInformation*.

INTERNET_STATUS_SENDING_REQUEST
Sending the information request to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_ REQUEST_SENT
Successfully sent the information request to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_RECEIVING_RESPONSE
Waiting for the server to respond to a request. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_RESPONSE_RECEIVED
Successfully received a response from the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_CLOSING_CONNECTION
Closing the connection to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_CONNECTION_CLOSED
Successfully closed the connection to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_HANDLE_CREATED
Used by InternetConnect to indicate it has created the new handle. This lets the application call InternetCloseHandle from another thread, if the connect is taking too

long.

INTERNET_STATUS_HANDLE_CLOSING

This handle value is now terminated

INTERNET_STATUS_REQUEST_COMPLETE

An asynchronous operation has been completed. See InternetOpen for details on INTERNET_FLAG_ASYNC.

The *lpvStatusInformation* parameter points to an INTERNET_ASYNC_RESULT structure. The *dwStatusInformationLength* parameter contains the final completion status of the asynchronous function. If this is ERROR_INTERNET_EXTENDED_ERROR, the application can retrieve the server error information by using InternetGetLastResponseInfo.

lpvStatusInformation

Address of a buffer that contains information pertinent to this call to the callback function.

dwStatusInformationLength

Size of the *lpvStatusInformation* buffer.

Return Values

No return value.

Remarks

In the case of **INTERNET_STATUS_REQUEST_COMPLETE**, *lpvStatusInformation* is the address of an INTERNET_ASYNC_RESULT structure.

An application uses the callback function to indicate the progress of synchronous and asynchronous functions, and to indicate the completion of an asynchronous request.

Because callbacks are made during processing of the request, the application should spend as little time as possible in the callback function to avoid degrading data throughput on the network. For example, displaying a dialog box in a callback function may be such a lengthy operation that the server terminates the request.

The callback function may be called in a thread context different from the thread that initiated the request.

See Also

InternetCloseHandle, InternetConnect, InternetGetLastResponseInfo, InternetOpen

InternetTimeFromSystemTime

Formats a date and time according to the specified RFC format.

BOOL InternetTimeFromSystemTime(
IN CONST SYSTEMTIME **pst*,
IN DWORD *dwRFC*,
OUT LPSTR *lpszTime*,
IN DWORD *cbTime* **);**

Parameters

pst
Address of a **SYSTEMTIME** structure that contains the date and time to format.

dwRFC
RFC format.

lpszTime
Address of a buffer that receives the formatted data and time.

cbTime
Size, in bytes, of the *lpszTime* buffer.

Return Values

Returns **TRUE** if the function succeeds or **FALSE otherwise**. To get extended error information, call **GetLastError**.

InternetWriteFile

Writes data to an open Internet file.

BOOL InternetWriteFile(
IN HINTERNET *hFile*,
IN LPCVOID *lpBuffer*,
IN DWORD *dwNumberOfBytesToWrite*,
OUT LPDWORD *lpdwNumberOfBytesWritten* **);**

Parameters

hFile
Valid handle returned from a previous call to FtpOpenFile.

lpBuffer
Address of a buffer that contains the data to be written to the file.

dwNumberOfBytesToWrite
Number of bytes to write to the file.

lpdwNumberOfBytesWritten
Address of a variable that receives the number of bytes written to the buffer. The **InternetWriteFile** function sets this value to zero before doing any work or error checking.

Return Values

Returns **TRUE** if the function succeeds or **FALSE** otherwise. To get extended error information, call **GetLastError**. An application can also use InternetGetLastResponseInfo, when necessary.

Remarks

When the application is sending data, it must call InternetCloseHandle to end the data transfer.

See Also

FtpOpenFile, InternetCloseHandle

FTP Functions

The FTP functions deal with the FTP file and directory manipulation and navigation.

FtpCreateDirectory

Creates a new directory on the FTP server.

BOOL FtpCreateDirectory(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszDirectory* **);**

Parameters

hFtpSession
Calid handle to an FTP session.

lpszDirectory
Address of a null-terminated string that contains the name of the directory to create on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

Return Values

Return **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to create a directory, use InternetGetLastResponseInfo to determine why.

Remarks

An application should use FtpGetCurrentDirectory to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified filenames relative to the current directory. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpCreateDirectory** function translates the directory name separators to the appropriate character before they are used.

FtpDeleteFile

Deletes a file stored on the FTP server.

BOOL FtpDeleteFile(
  IN HINTERNET *hFtpSession*,
  IN LPCTSTR *lpszFileName* **);**

Parameters

hFtpSession
  Valid handle to an FTP session.

lpszFileName
  Address of a null-terminated string that contains the name of the file to delete on the remote system.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

The *lpszFile* parameter can be either partially qualified filenames relative to the current directory or fully qualified. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpDeleteFile** function translates the directory name separators to the appropriate character before they are used.

FtpFindFirstFile

Begins searching the current directory of the given FTP session. File and directory entries are returned to the application in the **WIN32_FIND_DATA** structure.

HINTERNET FtpFindFirstFile(
IN HINTERNET *hFtpSession*,
IN LPCTSTR lpszSearchFile *OPTIONAL*,
OUT LPWIN32_FIND_DATA *lpFindFileData*,
IN DWORD dwFlags IN DWORD *dwContext* **);**

Parameters

hFtpSession
Valid handle to an FTP session returned from InternetConnect.

lpszSearchFile
Address of a null-terminated string that specifies a valid directory path name or filename for the FTP server's file system.

lpFindFileData
Address of a **WIN32_FIND_DATA** structure that receives information about the found file or directory.

dwFlags
Application-defined value that associates this search with any application. For a description of the values, see InternetOpenUrl.

dwContext
An application-defined value that associates this search with any application data. This parameter is used only if the application has already called InternetSetStatusCallback to set up a status callback function.

Return Values

Returns a valid handle for the request, if the directory enumeration was started successfully; otherwise, returns NULL. To get a specific error code, call **GetLastError**. If the function finds no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

Remarks

This function enumerates both files and directories.

The **FtpFindFirstFile** function is similar to the Win32 FindFirstFile function. Note, however, an important difference is that only one **FtpFindFirstFile** can occur at a time within a given FTP session. The enumerations, therefore, are correlated with the FTP session handle. This is because the FTP protocol allows only a single directory enumeration per session.

After calling **FtpFindFirstFile** and until calling InternetCloseHandle, the application cannot call **FtpFindFirstFile** again on a given FTP session handle. If this happens, calls to the **FtpFindFirstFile** function will fail with error code ERROR_FTP_TRANSFER_IN_PROGRESS.

After beginning a directory enumeration with **FtpFindFirstFile**, the InternetFindNextFile function can be used to continue the enumeration.

The InternetCloseHandle function is used to close the handle returned from **FtpFindFirstFile**. If the InternetCloseHandle function closes the handle before InternetFindNextFile fails with ERROR_NO_MORE_FILES, the directory enumeration will be terminated.

Because the FTP protocol provides no standard means of enumerating, some of the common information about files, such as file creation date and file, is not always available or correct. When this happens, **FtpFindFirstFile** and InternetFindNextFile fill in unavailable information with a "best guess" based on available information. For example, creation and last access dates will often be the same as the file's modification date.

The application cannot call **FtpFindFirstFile** between calls to FtpOpenFile and InternetCloseHandle.

See Also

FtpOpenFile, InternetCloseHandle, InternetFindNextFile, InternetSetStatusCallback

FtpGetCurrentDirectory

Retrieves the current directory for the specified FTP session.

BOOL FtpGetCurrentDirectory(
IN HINTERNET *hFtpSession*,
OUT LPCTSTR *lpszCurrentDirectory*,
IN OUT LPDWORD *lpdwCurrentDirectory* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszCurrentDirectory
Address of a buffer that receives the current directory string, which specifies the absolute path to the current directory. The string is null terminated.

lpdwCurrentDirectory
Address of a variable that specifies the length, in characters, of the buffer for the current directory string. The buffer length must include room for a terminating null character. Using a length of **MAX_PATH** is sufficient for all path names. When the function returns, this parameter receives the number of characters copied into the buffer.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get the specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to change to a directory, use InternetGetLastResponseInfo to determine why.

Remarks

If the *lpszCurrentDirectory* buffer is not large enough, *lpdwCurrentDirectory* receives the number of bytes required to retrieve the full, current directory name.

FtpGetFile

Retrieves a file from the FTP server and stores it under the specified filename, creating a new local file in the process.

BOOL FtpGetFile(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszRemoteFile*,
IN LPCTSTR *lpszNewFile*,
IN BOOL *fFailIfExists*,
IN DWORD *dwFlagsAndAttributes*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszRemoteFile
Address of a null-terminated string that contains the name of the file to retrieve from the remote system.

lpszNewFile
Address of a null-terminated string that contains the name of the file to create on the local system.

fFailIfExists
Boolean flag that indicates whether the function should proceed if a local file of the specified name already exists. If *fFailIfExists* is **TRUE** and the local file exists, **FtpGetFile** fails.

dwFlagsAndAttributes
File attributes and flags for the new file. Can be any combination of FILE_ATTRIBUTE_* and INTERNET_FLAG_* file attributes. See **CreateFile** for further information on FILE_ATTRIBUTE_* attributes, and see InternetOpenUrl for further information on INTERNET_FLAG_* flags.

dwFlags
Flag value that indicates the conditions under which the transfer occurs. Can be any of the FTP_TRANSFER_TYPE_* values. For a description of these values, see FtpOpenFile.

dwContext
Application-defined value that associates this search with any application data. This is only used if the application has already called InternetSetStatusCallback to set up a status callback function.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

The **FtpGetFile** function is a high-level routine that handles all the bookkeeping and overhead associated with reading a file from an FTP server and storing it locally. An application that needs to retrieve file data only or that requires close control over the file transfer should use the FtpOpenFile and InternetReadFile functions.

If the *dwTransferType* specifies **FILE_TRANSFER_TYPE_ASCII**, translation of the file data converts control and formatting characters to local equivalents. The default transfer is binary mode where the file is downloaded in the same format as it is stored on the server.

Both *lpszRemoteFile* and *lpszNewFile* can be either partially qualified filenames relative to the current directory, or fully qualified. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpGetFile** function translates the directory name separators to the appropriate character before they are used.

FtpOpenFile

Initiates access to a remote file for writing or reading.

HINTERNET FtpOpenFile(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszFileName*,
IN DWORD *fdwAccess*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszFileName
Address of a null-terminated string that contains the name of the file to access on the remote system.

fdwAccess
Value that determines how the file will be accessed. This can be **GENERIC_READ** or **GENERIC_WRITE**, but not both.

dwFlags
Conditions under which subsequent transfers occur. Can be any of these values:

Value	Meaning
FTP_TRANSFER_TYPE_ASCII	Transfer the file using FTP's ASCII (Type A) transfer method. Control and formatting information is converted to local equivalents.
FTP_TRANSFER_TYPE_BINARY	Transfer the file using FTP's Image (Type I) transfer method. The file is transferred exactly as it exists with no changes. This is the default transfer method.

dwContext
Application-defined value that associates this search with any application data. This is only used if the application has already called InternetSetStatusCallback to set up a status callback function..

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

The **FtpOpenFile** function should be used in the following situations:

- An application has data it needs to send to an FTP server to be created as a file on the FTP server, but the application does not have a local file containing the data. After InternetOpen opens the file with **FtpOpenFile**, the application uses InternetWriteFile to send the FTP file data to the server.
- An application needs to retrieve a file from the server into application-controlled memory, instead of writing the file to disk. The application uses InternetReadFile after using InternetOpen to open the file.
- An application needs a fine level of control over a file transfer. For example, the application may need to display a "thermometer" when downloading a file to indicate to the user that the file transfer

is or is not proceeding correctly.

After calling the **FtpOpenFile** function and until calling InternetCloseHandle, the application can only call InternetReadFile orInternetWriteFile, InternetCloseHandle, or the FtpFindFirstFile function. Calls to other FTP functions on the same FTP session will fail and set the error code to ERROR_FTP_TRANSFER_IN_PROGRESS.

Only one file can be open in a single FTP session. Therefore, no file handle is returned and the application simply uses the FTP session handle when necessary.

The *lpszFile* parameter can be either partially qualified filenames relative to the current directory or fully qualified. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpOpenFile** function translates the directory name separators to the appropriate character before using it.

The InternetCloseHandle function is used to close the handle returned from **FtpOpenFile**. If the InternetCloseHandle function closes the handle before all the data has been transferred, the transfer is terminated.

FtpPutFile

Stores a file on the FTP server.

BOOL FtpPutFile(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszLocalFile*,
IN LPCTSTR *lpszNewRemoteFile*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszLocalFile
Address of a null-terminated string that contains the name of the file to send from the local system.

lpszNewRemoteFile
Address of a null-terminated string that contains the name of the file to create on the remote system.

dwFlags
Conditions under which the transfer occurs. Can be any combination of FTP_TRANSFER_* defined constants. For further information on the FTP_TRANSFER_* constants, see FtpOpenFile.

dwContext
Application-defined value that associates this search with any application data. This parameter is used only if the application has already called InternetSetStatusCallback to set up a status callback.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

The **FtpPutFile** function is a high-level routine that handles all the bookkeeping and overhead associated with reading a file from an FTP server and storing it locally. An application that needs to send file data only, or that requires close control over the file transfer, should use the FtpOpenFile and InternetWriteFile functions.

If the *dwTransferType* specifies **FILE_TRANSFER_TYPE_ASCII**, translation of the file data converts control and formatting characters to local equivalents.

Both *lpszNewRemoteFile* and *lpszLocalFile* can be either partially qualified filenames relative to the current directory, or fully qualified. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpPutFile** function translates the directory name separators to the appropriate character before they are used.

FtpRemoveDirectory

Removes the specified directory on the FTP server.

BOOL FtpRemoveDirectory(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszDirectory* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszDirectory
Address of a null-terminated string that contains the name of the directory to remove on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get the specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to remove a directory, use InternetGetLastResponseInfo to determine why.

Remarks

An application should use FtpGetCurrentDirectory to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified filenames relative to the current directory. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpRemoveDirectory** function translates the directory name separators to the appropriate character before they are used.

FtpRenameFile

Renames a file stored on the FTP server.

BOOL FtpRenameFile(
  IN HINTERNET *hFtpSession*,
  IN LPCTSTR *lpszExisting*,
  IN LPCTSTR *lpszNew* **);**

Parameters

hFtpSession
  Valid handle to an FTP session.

lpszExisting
  Address of a null-terminated string that contains the name of the file that will have its name changed on the remote FTP server.

lpszNew
  Address of a null-terminated string that contains the new name for the remote file.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get a specific error code, call **GetLastError**.

Remarks

The *lpszExisting* and *lpszNew* parameters can be either partially qualified filenames relative to the current directory or fully qualified. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpRenameFile** function translates the directory name separators to the appropriate character before they are used.

FtpSetCurrentDirectory

Changes to a different working directory on the FTP server.

BOOL FtpSetCurrentDirectory(
IN HINTERNET *hFtpSession*,
IN LPCTSTR *lpszDirectory* **);**

Parameters

hFtpSession
Valid handle to an FTP session.

lpszDirectory
Address of a null-terminated string that contains the name of the directory to change to on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

Return Values

Returns **TRUE** if successful or **FALSE otherwise**. To get the specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to change a directory, use InternetGetLastResponseInfo to determine why.

Remarks

An application should use FtpGetCurrentDirectory to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified filenames relative to the current directory. A backslash ("\") or forward slash ("/") can be used as the directory separator for either name. The **FtpSetCurrentDirectory** function translates the directory name separators to the appropriate character before they are used.

Gopher Functions

The Gopher functions control the creation and use of the Internet Gopher utilities.

GopherCreateLocator

Creates a Gopher or Gopher+ locator string from its component parts.

BOOL GopherCreateLocator(
IN LPCTSTR *lpszHost*,
IN INTERNET_PORT *nServerPort*,
IN LPCTSTR lpszDisplayString *OPTIONAL*,
IN LPCTSTR lpszSelectorString *OPTIONAL*,
IN DWORD *dwGopherType*,
OUT LPCTSTR lpszLocator *OPTIONAL*,
IN OUT LPDWORD *lpdwBufferLength* **);**

Parameters

lpszHost
Address of a string that contains the name of the host, or a dotted-decimal IP address (such as 198.105.232.1).

nServerPort
Number of the port on which the Gopher server at *lpszHost* lives, in host byte order. If *nPort* is INTERNET_INVALID_PORT_NUMBER, the default Gopher port is read from the \etc\services file.

lpszDisplayString
Gopher document or directory to be displayed. If this parameter is NULL, the function returns the default directory for the Gopher server.

lpszSelectorString
Address of the selector string to send to the Gopher server in order to retrieve information. This parameter can be NULL.

dwGopherType
Value that specifies whether *lpszSelector* refers to a directory or document, and whether the request is Gopher+ or Gopher. For more information, see GOPHER_FIND_DATA.

lpszLocator
Address of a buffer that receives the locator string. If *lpszLocator* is NULL, *lpdwBufferLength* receives the needed buffer length, but the function performs no other processing.

lpdwBufferLength
Length of the *lpszLocator* buffer. When the function returns, this parameter receives the number of bytes written to the *lpszLocator* buffer. If **GetLastError** returns **ERROR_INSUFFICIENT_BUFFER**, this parameter receives the number of bytes required to form the locator successfully.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get the extended error information, call **GetLastError** or InternetGetLastResponseInfo.

Remarks

To retrieve information from a Gopher server, an application must first get a Gopher "locator" from the Gopher server.

The locator, which the application should treat as an opaque token, is normally used for calls to the GopherFindFirstFile function to retrieve a specific piece of information.

GopherGetLocatorType

Parses a Gopher locator and determines its attributes.

BOOL GopherGetLocatorType(
IN LPCTSTR *lpszLocator*,
OUT LPDWORD *lpdwGopherType* **);**

Parameters

lpszLocator
Address of the Gopher locator string to parse.

lpdwGopherType
Address of a variable that receives the type of the locator. The type is a bitmask that consists of a combination of the following values:

Value	Meaning
GOPHER_TYPE_TEXT_FILE	An ASCII text file.
GOPHER_TYPE_DIRECTORY	A directory of additional Gopher items.
GOPHER_TYPE_CSO	A CSO telephone book server.
GOPHER_TYPE_ERROR	Indicates an error condition.
GOPHER_TYPE_MAC_BINHEX	A Macintosh file in BINHEX format.
GOPHER_TYPE_DOS_ARCHIVE	An MS-DOS archive file.
GOPHER_TYPE_UNIX_UUENCODED	A UUENCODED file.
GOPHER_TYPE_INDEX_SERVER	Refers to an index server.
GOPHER_TYPE_TELNET	A Telnet Server.
GOPHER_TYPE_BINARY	A binary file.
GOPHER_TYPE_REDUNDANT	Refers to a duplicated server. The information contained within is a duplicate of the primary server. The primary server is defined as the last directory entry that did not have a **GOPHER_TYPE_REDUNDANT** type.
GOPHER_TYPE_TN3270	A TN3270 server.
GOPHER_TYPE_GIF	A GIF graphics file.
GOPHER_TYPE_IMAGE	An image file.
GOPHER_TYPE_BITMAP	A bitmap file.
GOPHER_TYPE_MOVIE	A movie file.
GOPHER_TYPE_SOUND	A sound file.
GOPHER_TYPE_HTML	An HTML document.
GOPHER_TYPE_PDF	A PDF file.
GOPHER_TYPE_CALENDAR	A calendar file.
GOPHER_TYPE_INLINE	An inline file.
GOPHER_TYPE_UNKNOWN	The item type is unknown.
GOPHER_TYPE_ASK	An Ask+ item.
GOPHER_TYPE_GOPHER_PLUS	A Gopher+ item.

Remarks

The **GopherGetLocatorType** function returns information about the item referenced by a Gopher locator. Note that it is possible for multiple attributes to be set on a file. For example, both **GOPHER_TYPE_TEXT_FILE** and **GOPHER_TYPE_GOPHER_PLUS** are set for a text file stored on a Gopher+ server.

GopherFindFirstFile

Uses a Gopher locator and some search criteria to create a session with the server and locate the requested documents, binary files, index servers, or directory trees.

HINTERNET GopherFindFirstFile(
IN HINTERNET *hGopherSession*,
IN LPCTSTR lpszLocator *OPTIONAL*,
IN LPCTSTR lpszSearchString *OPTIONAL*,
OUT LPGOPHER_FIND_DATA lpFindData *OPTIONAL*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hGopherSession
Handle to a Gopher session returned by InternetConnect.

lpszLocator
Name of the item to locate. Can be one of the following items:

- A Gopher locator returned by *lpGopherFindData*, or a locator obtained by a previous call to this function or the InternetFindNextFile function.
- A NULL pointer or zero-length string indicating that the topmost information from a Gopher server is being returned.
- A locator created by the GopherCreateLocator function.

lpszSearchString
Address of a buffer that contains the strings to search, it this request is to an index server. Otherwise, this parameter should be NULL.

lpFindData
Address of a GOPHER_FIND_DATA structure that receives the information retrieved by this function.

dwFlags
Action flag. For a list of the valid flag values, see InternetOpenUrl.

dwContext
Application-defined value that associates this search with any application data.

Return Values

Returns a valid search handle if successful, or NULL otherwise. To get extended error information, call **GetLastError** or InternetGetLastResponseInfo.

Remarks

The **GopherFindFirstFile** function closely resembles the Win32 function FindFirstFile. It creates a connection with a Gopher server and then returns a single structure containing information about the first Gopher object referenced by the locator string.

After calling **GopherFindFirstFile** to retrieve the first Gopher object in an enumeration, an application can use the InternetFindNextFile function to retrieve subsequent Gopher objects.

Use the InternetCloseHandle function to close the handle returned from **GopherFindFirstFile**. If there are any pending operations described by the handle when the application calls

InternetCloseHandle, they are canceled or marked closed pending. Any open sessions are terminated and any data waiting for the caller is discarded. In addition, any allocated buffers are freed.

See Also

InternetCloseHandle, InternetConnect, InternetFindNextFile

GopherGetAttribute

Allows an application to retrieve specific attribute information from the server.

BOOL GopherGetAttribute(
IN HINTERNET *hGopherSession*,
IN LPCTSTR *lpszLocator*,
IN LPCTSTR lpszAttributeName *OPTIONAL*,
OUT LPBYTE *lpBuffer*,
IN DWORD *dwBufferLength*,
OUT LPDWORD *lpdwCharactersReturned*,
IN GOPHER_ATTRIBUTE_ENUMERATOR lpfnEnumerator *OPTIONAL*,
IN DWORD *dwContext* **);**

Parameters

hGopherSession
Handle to a Gopher session returned by InternetConnect.

lpszLocator
Address of a string that identifies the item at the Gopher server on which to return attribute information.

lpszAttributeName
Address of a space-delimited string specifying the names of attributes to return.

lpBuffer
Address of an application-defined buffer from which attribute information is retrieved.

dwBufferLength
Size, in bytes, of the *lpBuffer* buffer.

lpdwCharactersReturned
Number of characters read into the *lpBuffer* buffer.

lpfnEnumerator
Address of a callback function that enumerates each attribute of the locator. This parameter is optional. If it is NULL, all the Gopher attribute information is placed into *lpBuffer*. If *lpfnEnumerator* is specified, the callback function is called once for each attribute of the object.

The callback function receives the address of a single GOPHER_ATTRIBUTE_TYPE structure with each call. The enumeration callback function allows the application to avoid having to parse the Gopher attribute information.

dwContext
Application-defined value that associates this operation with any application data.

Return Values

Returns **TRUE** if the request is satisfied or **FALSE** otherwise. To get extended error information, call **GetLastError** or InternetGetLastResponseInfo.

Remarks

Generally, applications call this function after calling GopherFindFirstFile or InternetFindNextFile.

The size of the *lpBuffer* parameter must be equal to or greater than the **MIN_GOPHER_ATTRIBUTE_LENGTH** (currently defined in WININET.H as 256 bytes).

See Also

InternetConnect

GopherAttributeEnumerator

Defines a callback function that processes attribute information from a Gopher server. This callback function is installed by a call to the GopherGetAttribute function.

BOOL GopherAttributeEnumerator(
LPGOPHER_ATTRIBUTE_TYPE *lpAttributeInformation*,
DWORD *dwError* **);**

Parameters

lpAttributeInformation
Address of a buffer that contains a single GOPHER_ATTRIBUTE_TYPE structure. The *lpBuffer* parameter to GopherGetAttribute is used for storing this structure. The *lpBuffer* size must be equal to or greater than the value of **MIN_GOPHER_ATTRIBUTE_LENGTH**.

dwError
Error value. This parameter is **NO_ERROR** if the attribute was parsed and written to the buffer successfully. If a problem was encountered, an error value is returned.

Return Values

Return **TRUE** to continue the enumeration, or **FALSE** to immediately stop it. This function is primarily used for returning the results of a Gopher+ ASK item.

GopherOpenFile

Begins reading a Gopher data file from a Gopher server.

HINTERNET GopherOpenFile(
IN HINTERNET *hGopherSession*,
IN LPCTSTR *lpszLocator*,
IN LPCTSTR lpszView *OPTIONAL*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hGopherSession
Handle to a Gopher session returned by InternetConnect.

lpszLocator
Address of a string that identifies the file to open. Generally, this locator is returned from a call to GopherFindFirstFile or InternetFindNextFile. Because the Gopher protocol has no concept of a current directory, the locator is always fully qualified.

lpszView
Address of a string that describes the view to open if several views of the file exist at the server. If *lpszView* is NULL, the function uses the default file view.

dwFlags
Any combination of the INTERNET_FLAG_* flag values. For a list of valid flag values, see InternetOpenUrl.

dwContext
Application-defined value that associates this operation with any application data.

Return Values

Returns a handle if successful or NULL if the file cannot be opened. To get extended error information, call **GetLastError** or InternetGetLastResponseInfo.

Remarks

This function opens a file at a Gopher server. Because a file cannot be actually opened or locked at a server, this function simply associates location information with a handle that an application can use for file-based operations such as InternetReadFile or GopherGetAttribute.

Use the InternetCloseHandle function to close the handle returned from **GopherOpenFile**. If there are any pending operations described by the handle when the application calls InternetCloseHandle, they are canceled or marked closed pending. Any open sessions are terminated, and any data waiting for the caller is discarded. In addition, any allocated buffers are freed.

See Also

GopherFindFirstFile, GopherGetAttribute, InternetCloseHandle, InternetConnect, InternetFindNextFile, InternetOpenUrl, InternetReadFile

HTTP Functions

The HTTP functions control the transmission and content of HTTP requests.

HttpAddRequestHeaders

Adds one or more HTTP request headers to the HTTP request handle.

BOOL HttpAddRequestHeaders(
IN HINTERNET *hHttpRequest*,
IN LPCTSTR *lpszHeaders*,
IN DWORD *dwHeadersLength*,
IN DWORD *dwModifiers* **);**

Parameters

hHttpRequest
Open HTTP request handle returned by HttpOpenRequest.

lpszHeaders
Headers to append to the request. Each header must be terminated by a CR/LF (carriage return, line feed) pair.

dwHeadersLength
Length, in characters, of *lpszHeaders*. If this parameter is -1L, the function assumes that *lpszHeaders* is zero terminated (ASCIIZ) and the length is computed.

dwModifiers
Values used to modify the semantics of this function. Can be a combination of these values:

Modifier	Meaning
HTTP_ADDREQ_FLAG_COALESCE	Coalesces headers of the same name. For example, adding "Accept: text/*" followed by "Accept: audio/*" with this flag results in the formation of the single header "Accept: text/*, audio/*". This causes the first header found to be coalesced. It is up to the calling application to ensure a cohesive scheme with respect to coalesced /separate headers.
HTTP_ADDREQ_FLAG_REPLACE	Performs a remove + add. The header name is used to remove the current header, and the full value is used to add the new header. ADDREQ_COALESCE is meaningful. Used to replace or remove a header. If the header-value is empty and the header is found, it is removed. If not empty, the header-value is replaced.
HTTP_ADDREQ_FLAG_ADD_IF_NEW	Adds the header only if it does not already exist; otherwise, an error is returned.
HTTP_ADDREQ_FLAG_ADD	Used with REPLACE. Adds the header if it doesn't exist.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

Remarks

This function appends additional, free-format headers to the HTTP request handle and is intended for use by sophisticated clients that need detailed control over the exact request sent to the HTTP server.

Note that for basic **HttpAddRequestHeaders**, the application can pass in multiple headers in a single buffer. If the application is trying to remove or replace a header, only one header can be supplied in *lpszHeaders.*

See Also

HttpOpenRequest, HttpSendRequest

HttpOpenRequest

Opens an HTTP request handle.

HINTERNET HttpOpenRequest(
IN HINTERNET *hHttpSession*,
IN LPCTSTR *lpszVerb*,
IN LPCTSTR *lpszObjectName*,
IN LPCTSTR *lpszVersion*,
IN LPCTSTR lpszReferer *OPTIONAL*,
IN LPCTSTR FAR * lpszAcceptTypes *OPTIONAL*,
IN DWORD *dwFlags*,
IN DWORD *dwContext* **);**

Parameters

hHttpSession
Handle to an HTTP session returned by InternetConnect.

lpszVerb
Address of a string that contains the verb to use in the request. If this parameter is NULL, the function uses "GET" as the verb.

lpszObjectName
Address of a string that contains the name of the target object of the specified verb. This is generally a filename, an executable module, or a search specifier.

lpszVersion
Address of a string that contains the HTTP version. If this parameter is NULL, the function uses "HTTP/1.0" as the version.

lpszReferer
Address of a string that specifies the address (URL) of the document from which the URL in the request (*lpszObjectName*) was obtained. If this parameter is NULL, no "referer" is specified.

lpszAcceptTypes
Address of a null-terminated array of LPCTSTR pointers indicating content types accepted by the client. If this parameter is NULL, no types are accepted by the client. Servers interpret a lack of accept types to indicate that the client only accepts documents of type "text/*" (that is, only text documents and not pictures or other binary files).

dwFlags
Internet flag values. For a list of valid flag values, see InternetOpenUrl.

dwContext
An application-defined value that associates this operation with any application data.

Return Values

Returns a valid (non-NULL) HTTP request handle if successful, or NULL otherwise. To get extended error information, call **GetLastError**.

Remarks

This function creates a new HTTP request handle and stores the specified parameters in that handle. An HTTP request handle holds a request to be sent to an HTTP server and contains all RFC822/MIME/HTTP headers to be sent as part of the request.

Use the InternetCloseHandle function to close the handle returned by **HttpOpenRequest**. The

InternetCloseHandle cancels all outstanding I/O on the handle.

The *lpszCallerApplicationName* parameter to InternetOpen is used as the referrer for the HTTP request.

See Also

HttpAddRequestHeaders, HttpQueryInfo, HttpSendRequest, InternetCloseHandle, InternetConnect, InternetOpen, InternetReadFile

HttpQueryInfo

Queries for information about an HTTP request.

BOOL HttpQueryInfo(
IN HINTERNET *hHttpRequest*,
IN DWORD *dwInfoLevel*,
IN LPVOID lpvBuffer *OPTIONAL*,
IN LPWORD *lpvBufferLength*,
IN OUT LPWORD lpdwIndex *OPTIONAL*,**);**

Parameters

hHttpRequest
Open HTTP request handle returned by HttpOpenRequest.

dwInfoLevel
Combination of the attribute to query and flags that modify the request. In addition to the HTTP_QUERY_ *values described below, the application can use the following flags to modify the request:

HTTP_QUERY_CUSTOM
If this query level is specified, *lpvBuffer* contains an ASCIIZ header name. This header name is searched for and its value returned in *lpvBuffer* on output.

HTTP_QUERY_FLAG_COALESCE
Combine the values from several headers of the same name into the output buffer.

HTTP_QUERY_FLAG_REQUEST_HEADERS
Typically, response headers are queried, but an application can also query request headers by using this flag.

HTTP_QUERY_FLAG_SYSTEMTIME
For those headers whose value is a date/time string, such as "Last-Modified-Time", specifying this flag returns the header-value as a standard Win32 **SYSTEMTIME** structure, which does not require the application to parse the data.

HTTP_QUERY_FLAG_NUMBER
For those headers whose value is a number, such as the status code, specifying this flag returns the data as a 32-bit number.

lpvBuffer
[In/Out] Address of the buffer that receives the information.

lpdwBufferLength
[In/Out] Address of a value that contains the length of the data buffer. When the function returns, this parameter contains the address of a value specifying the length of the information written to the buffer. When the function returns strings, the following rules apply:

- If the function succeeds, *lpdwBufferLength* specifies the length of the string, in characters, minus one for the terminating null.
- If the function fails and **ERROR_INSUFFICIENT_BUFFER** is returned, *lpdwBufferLength* specifies the number of bytes that the application must allocate in order to receive the string.

lpdwIndex

[In/Out] Address of a zero-based header index used to enumerate multiple headers with the same name. When calling the function, this parameter is the index of the specified header to return. When the function returns, this parameter is the index of the next header. If the next index cannot be found, ERROR_HTTP_HEADER_NOT_FOUND is returned.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

Remarks

The possible values for the *dwInfoLevel* parameter include:

HTTP_QUERY_MIME_VERSION
HTTP_QUERY_CONTENT_TYPE
HTTP_QUERY_CONTENT_TRANSFER_ENCODING
HTTP_QUERY_CONTENT_ID
HTTP_QUERY_CONTENT_DESCRIPTION
HTTP_QUERY_CONTENT_LENGTH
HTTP_QUERY_ALLOW
HTTP_QUERY_PUBLIC
HTTP_QUERY_DATE
HTTP_QUERY_EXPIRES
HTTP_QUERY_LAST_MODIFIED
HTTP_QUERY_MESSAGE_ID
HTTP_QUERY_URI
HTTP_QUERY_DERIVED_FROM
HTTP_QUERY_LANGUAGE
HTTP_QUERY_COST
HTTP_QUERY_WWW_LINK
HTTP_QUERY_PRAGMA
HTTP_QUERY_VERSION
HTTP_QUERY_STATUS_CODE
HTTP_QUERY_STATUS_TEXT
HTTP_QUERY_RAW_HEADERS
HTTP_QUERY_RAW_HEADERS_CRLF
HTTP_QUERY_REQUEST_METHOD The *lpvBuffer* parameter receives the verb that is being used in the request, typically "GET" or "POST".

This function is used to return response or request headers from a HTTP request. You can retrieve different types of data from **HttpQueryInfo**:

- strings (default)
- **SYSTEMTIME** (for Data: Expires: etc, headers)
- DWORD (for STATUS_CODE, CONTENT_LENGTH, etc.)

See Also

HttpOpenRequest

HttpSendRequest

Sends the specified request to the HTTP server.

BOOL HttpSendRequest(
IN HINTERNET *hHttpRequest*,
IN LPCTSTR lpszHeaders *OPTIONAL*,
IN DWORD *dwHeadersLength*,
IN LPVOID lpOptional *OPTIONAL*,
DWORD *dwOptionalLength* **);**

Parameters

hHttpRequest
Open HTTP request handle returned by HttpOpenRequest.

lpszHeaders
Additional headers to be appended to the request. This parameter can be NULL if there are no additional headers to append.

dwHeadersLength
Length, in characters, of the additional headers. If this parameter is -1L and *lpszHeaders* is not NULL, the function assumes that *lpszHeaders* is zero terminated (ASCIIZ) and the length is calculated.

lpOptional
Address of any optional data to send immediately after the request headers. This is generally used for POST and PUT operations. This parameter can be NULL if there is no optional data to send.

dwOptionalLength
Length, in bytes, of the optional data. This parameter can be zero if there is no optional data to send.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

Remarks

This function sends the specified request to the HTTP server and allows the client to specify additional RFC822/MIME/HTTP headers to send along with the request.

The function also lets the client specify optional data to send to the HTTP server immediately following the request headers. This feature is generally used for "write" operations such as PUT and POST.

After the request is sent, the status code and response headers from the HTTP server are read. These headers are maintained internally and are available to client applications through the HttpQueryInfo function.

An application can use the same HTTP request handle in multiple calls to **HttpSendRequest**, but the application must read all data returned from the previous call before calling the function again.

See Also

HttpOpenRequest, HttpQueryInfo, InternetReadFile

Cookie Functions

Cookies are a means by which, under HTTP protocol, a server or a script can maintain state information on the client workstation. WinInet has implemented a persistent cookie database for this purpose. Cookie functions are provided for users of WinInet in order to set cookies into, and access them from, the cookie database. The caller of these functions should be familiar with cookies as outlined in ftp://ds.internic.net/internet-drafts/draft-ietf-http-state-mgmt-01.txt. Please note that the implementation of these functions is evolving; be cautious when using them.

InternetGetCookie

Returns cookies for the specified URL and all it's parent URLs.

BOOL InternetGetCookie(
  IN LPCSTR *lpszUrlName*,
  IN LPCSTR *lpszCookieName*,
  OUT LPSTR *lpszCookieData*,
  IN OUT LPDWORD *lpdwSize* **);**

Parameters

lpszUrlName
  Address of a string that contains the URL to get cookies for.

lpszCookieName
  Address of the name of the cookie to get for the specified URL. This has not been implemented in this release.

lpszCookieData
  Address of the buffer that receives the cookie data. This value can be NULL.

lpdwSize
  Address of a variable that specifies the size of the *lpszCookieData* buffer. If the function succeeds, the buffer receives the amount of data copied to the *lpszCookieData* buffer. If *lpCookieData* is NULL, this parameter receives a value that specifies the size of buffer necessary to copy all the cookie data.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get the specific error value, call **GetLastError**. The following error values apply to **InternetGetCookie**:

Value	Description
ERROR_FILE_NOT_FOUND	There is no cookie for the specified URL and all it's parents.
ERROR_INSUFFICIENT_BUFFER	The value passed in *lpdwSize* is insufficient to copy all the cookie data. The value returned in *lpdwSize* is the size of buffer necessary to get all the data.

InternetSetCookie

Sets a cookie on the specified URL.

BOOL InternetSetCookie(
  IN LPCSTR *lpszUrlName***,**
  IN LPCSTR *lpszCookieName***,**
  IN LPCSTR *lpszCookieData* **);**

Parameters

lpszUrlname
  Address of a null-terminated string that specifies the URL for which the cookie should be set.

lpszCookieName
  Address of a string that contains the name to associate with the cookie. If this parameter is NULL, no name is associated with the cookie. This parameter is not implemented in this release and should be set to NULL.

lpszCookieData
  Address of the actual data to associate with the URL.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get the specific error code, call **GetLastError**.

Persistent URL Cache Functions

This section describes the functions used by clients that need persistent caching services. The functions allow an application to save data in the local file system for subsequent use, such as in situations where access to the data is over a low bandwidth link, or the access is not available at all. The calling program that inserts data into the persistent cache assigns a source name that is used to do operations like retrieve the data, set and get some properties on it, and delete it.

The protocols implemented in WinInet use the cache functions to provide persistent caching and off-line browsing. Unless explicitly specified not to cache through the INTERNET_FLAG_NO_CACHE_WRITE flag, WinInet caches all data downloaded from the net. Caveat: As of 5/15/96 the responses to POST data are not cached, but this will be added in the next release.

Handling Structures with Variable Size Information

The cache may contain variable size information for each URL stored. This is reflected in the INTERNET_CACHE_ENTRY_INFO structure. When the cache functions return this structure, they create a buffer which is always the size of INTERNET_CACHE_ENTRY_INFO plus any variable-size information. If a pointer member is not NULL, it points to the memory area immediately after the structure. While copying the returned buffer from a function into another buffer, the pointer members should be fixed up to point to the appropriate place in the new buffer as the following example shows:

```
lpDstCEInfo->lpszSourceUrlName =
    (LPINTERNET_CAHCE_ENTRY_INFO) ((LPBYTE) lpSrcCEInfo +
((DWORD) (lpOldCEInfo->lpszSourceUrlName) - (DWORD) lpOldCEInfo))
```

Some cache functions fail with the **ERROR_INSUFFICIENT_BUFFER** error value if you specify a buffer that is too small to contain the cache-entry information retrieved by the function. In this case, the function also returns the required size of the buffer. You can then allocate a buffer of the appropriate size and call the function again. If you want the function to succeed on the first call, allocate a buffer of the size specified by the MAX_CACHE_ENTRY_INFO_SIZE value, and then set the **dwStructSize** member of the INTERNET_CACHE_ENTRY_INFO structure to MAX_CACHE_ENTRY_INFO_SIZE when calling the function.

CommitUrlCacheEntry

Caches data in the specified file in the cache storage and associates it with the given URL.

BOOL CommitUrlCacheEntry(
IN LPCSTR *lpszUrlName*,
IN LPCTSTR *lpszLocalFileName*,
IN FILETIME *ExpireTime*,
IN FILETIME *LastModifiedTime*,
IN DWORD *CacheEntryType*,
IN LPCBYTE *lpHeaderInfo*,
IN DWORD *dwHeaderSize*,
IN LPCTSTR *lpszFileExtension*,
IN DWORD *dwReserved* **);**

Parameters

lpszUrlName
Address of a string that contains the source name of the cache entry. The name string must be unique, and should not contain any escape characters.

lpszLocalFileName
Address of a string that contains the name of the local file that is being cached. This should be the same name as that returned by CreateUrlCacheEntry.

ExpireTime
A **FILETIME** structure that contains the expire date and time (GMT) of the file that is being cached. If the expire date and time is unknown, set this parameter to zero.

LastModifiedTime
A **FILETIME** structure that contains the last modified date and time (GMT) of the URL that is being cached. If the last modified date and time is unknown, set this parameter to zero.

CacheEntryType
Type of the cache entry.

lpHeaderInfo
Address of the header information. If this parameter is not NULL, the header information is treated as extended attributes of the URL and is returned back in the INTERNET_CACHE_ENTRY_INFO structure.

dwHeaderSize
Size of the header information. If *lpHeaderInfo* is not NULL, this value is assumed to indicate the size of the header information. An application can maintain headers as part of the data and provide *dwHeaderSize*, together with a NULL value for *lpHeaderInfo*.

lpszFileExtension
Address of a buffer that contains information maintained in the cache database for future use. In this version of WinInet, this information is not used.

dwReserved
Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning

ERROR_FILE_NOT_FOUND	The specified local file is not found.
ERROR_DISK_FULL	The cache storage is full.

Remarks

If the cache storage is full, the function invokes cache cleanup to make space for this new file. If the file size is bigger than the cache size, the function returns **ERROR_DISK_FULL**. If the cache entry already exists, the function overwrites the entry. The user could specify SPARSE_CACHE_ENTRY in the Commit to indicate to himself that the size of the data is incomplete.

CreateUrlCacheEntry

Allocates requested cache storage and creates a local filename for saving the cache entry corresponding to the source name.

BOOL CreateUrlCacheEntry(
IN LPCSTR *lpszUrlName*,
IN DWORD *dwExpectedFileSize*,
IN DWORD *lpszFileExtension*,
OUT LPTSTR *lpszFileName*,
IN DWORD *dwReserved* **);**

Parameters

lpszUrlName
Address of a string that contains the name of the URL. The string should not contain any escape character.

dwExpectedFileSize
Expected size of the file needed to store the data corresponding to the source entity. If the expected size is unknown, set this value to zero.

lpszFileExtension
Address of a string that contains an extension to name of the file in the local storage.

lpszFileName
Address of a buffer that receives the filename. The buffer should be large enough (MAX_PATH) to store the file path name of the created file.

dwReserved
Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

Remarks

Internet services that use the cache should call this function to write directly into the cache storage. The caller should indicate the expected size of the file, but it is not guaranteed. Once the file is completely received, the caller should call CommitUrlCacheEntry to commit the entry in the cache.

GetUrlCacheEntryInfo

Retrieves information about a cache entry.

BOOL GetUrlCacheEntryInfo(
IN LPCSTR *lpszUrlName*,
IN LPINTERNET_CACHE_ENTRY_INFO *lpCacheEntryInfo*,
IN OUT LPDWORD *lpdwCacheEntryInfoBufferSize* **);**

Parameters

lpszUrlName
Address of a string that contains the name of the cache entry. The name string should not contain any escape character.

lpCacheEntryInfo
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about cache entry.

lpdwCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpszCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_INSUFFICIENT_BUFFER	The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information
ERROR_FILE_NOT_FOUND	Ihe specified cache entry is not found in the cache.

ReadUrlCacheEntryStream

Reads the cached data from a stream that has been opened using the RetrieveUrlCacheEntryStream function.

BOOL ReadUrlCacheEntryStream(
IN HANDLE *hUrlCacheStream*,
IN DWORD *dwLocation*,
IN OUT LPVOID *lpBuffer*,
IN OUT LPDWORD *lpdwLen*,
IN DWORD *dwReserved* **);**

Parameters

hUrlCacheStream
Handle that was returned by the RetrieveUrlCacheEntryStream function.

dwLocation
Offset to read from.

lpBuffer
Address of a buffer that receives the data.

lpdwLen
Address of a variable that specifies the length of the *lpBuffer* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

dwReserved
Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_INSUFFICIENT_BUFFER	The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information

RetrieveUrlCacheEntryFile

Retrieves a cache entry from the cache in the form of a file.

BOOL RetrieveUrlCacheEntryFile(
IN LPCSTR *lpszUrlName***,**
OUT LPINTERNET_CACHE_ENTRY_INFO *lpCacheEntryInfo***,**
IN OUT LPDWORD lpdwCacheEntryInfoBufferSize IN DWORD *dwReserved* **);**

Parameters

lpszUrlName
Source name of the cache entry. This must be an unique name. The name string should not contain any escape characters.

lpCacheEntryInfo
Address of a cache entry information buffer. If the buffer is not sufficient to accommodate all the information associated with the URL, one or more of the embedded pointers will be NULL.

lpdwCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpCacheEntryInfo* buffer. When the function returns, this variable contains the size of the actual buffer used or required. The caller should check the return value in this variable. If the return size is less than or equal to the size passed in, all the relevant data has been returned.

dwReserved
Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_FILE_NOT_FOUND	The cache entry specified by the source name is not found in the cache storage.
ERROR_INSUFFICIENT_BUFFER	The size of the *lpCacheEntryInfo* buffer as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all of the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to get all the information

Remarks

If an extension was provided while calling CommitUrlCacheEntry, the file will have the specified extension. If the entry is not available in the cache, this function returns **ERROR_FILE_NOT_FOUND**; otherwise, it returns the name of the local file. The caller is given only read permission to the local file, so the caller should not attempt to write or delete the file.

The file is locked for the caller when it is retrieved, the caller should unlock the file after it has been used up. Cache manager will automatically unlock the locked files after a certain interval. While the file is locked, cache manager will not remove the file from the cache. It is important to note that this function may be efficient or expensive depending on the internal implementation of the cache. For instance, if the URL data is stored in a packed file that contains data for other URLs, the cache will make a copy of the data to a file in a temporary directory maintained by the cache. The cache will eventually delete the copy. It is recommended that this function be used only in situations where a

filename is needed in order to launch an application. RetrieveUrlCacheEntryStream and associated stream functions should be used in most cases.

RetrieveUrlCacheEntryStream

Provides the most efficient and implementation independent way of accessing the cache data.

HANDLE RetrieveCacheEntryStream(
IN LPCSTR *lpszUrlName*,
OUT LPINTERNET_CACHE_ENTRY_INFO *lpEntryInfo*,
IN OUT LPDWORD *lpdwCacheEntryInfoBufferSize*,
IN BOOL *fRandomRead*,
IN DWORD *dwReserved* **);**

Parameters

lpszUrlName
Address of a string that contains the source name of the cache entry. This must be an unique name. The name string should not contain any escape characters.

lpCacheEntryInfo
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about the cache entry.

lpdwCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpCacheEntryInfo* buffer. When the function returns, the variable receives the number of bytes copied to the buffer, or the required size of the buffer.

fRandomRead
Flag to indicate that the stream is opened for random access or not. Set the flag to **TRUE** to open the stream for random access.

dwReserved
Reserved for future use.

Return Values

Returns a valid handle for use in the other stream function if successful, or **INVALID_HANDLE_VALUE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_FILE_NOT_FOUND	The cache entry specified by the source name is not found in the cache storage.
ERROR_INSUFFICIENT_BUFFER	The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information

Remarks

Cache clients that do not need URL data in the form of a file should use this function to access the data for a particular URL.

SetUrlCacheEntryInfo

Sets the specified members of the INTERNET_CACHE_ENTRY_INFO structure.

BOOL SetUrlCacheEntryInfo(
IN LPCSTR *lpszUrlName*,
IN LPINTERNET_CACHE_ENTRY_INFO *lpCacheEntryInfo*,
IN DWORD *dwFieldControl* **);**

Parameters

lpszUrlName
Address of a string that contains the name of the cache entry. The name string should not contain any escape character.

lpCacheEntryInfo
Address of a INTERNET_CACHE_ENTRY_INFO structure.

dwFieldControl
Bitmask that indicates the members that are to be set. Can be a combination of the following values:

CACHE_ENTRY_ATTRIBUTE_FC
CACHE_ENTRY_HITRATE_FC
CACHE_ENTRY_MODTIME_FC
CACHE_ENTRY_EXPTIME_FC
CACHE_ENTRY_ACCTIME_FC
CACHE_ENTRY_SYNCTIME_FC
CACHE_ENTRY_HEADERINFO_FC

The last two flags have not been implemented in this release.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_FILE_NOT_FOUND	The specified cache entry is not found in the cache.
ERROR_INVALID_PARAMETER	The value(s) to be set are invalid.

UnlockUrlCacheEntryFile

Unlocks the cache entry that was locked while the file was retrieved for use from the cache.

BOOL UnlockUrlCacheEntryFile(
  IN LPCSTR *lpszUrlName*,
  IN DWORD *dwReserved* **);**

Parameters

lpszUrlName
  Address of a string that contains the source name of the cache entry that is being unlocked. The name string should not contain any escape characters.

dwReserved
  Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_FILE_NOT_FOUND	The cache entry specified by the source name is not found in the cache storage.

Remarks

The application should not access the file after calling this function.

When this function returns, the cache manager is free to delete the cache entry.

UnlockUrlCacheEntryStream

Closes the stream that has been retrieved using the RetrieveUrlCacheEntryStream function.

BOOL UnlockUrlCacheEntryStream(
  IN HANDLE *hUrlCacheStream*,
  IN DWORD *dwReserved* **);**

Parameters

hUrlCacheStream
  Handle that was returned by the RetrieveUrlCacheEntryStream function.

dwReserved
  Reserved for future use.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

See Also

RetrieveUrlCacheEntryStream

DeleteUrlCacheEntry

Removes the file associated with the source name from the cache, if the file exists.

BOOL DeleteUrlCacheEntry(
IN LPCSTR *lpszUrlName* **);**

Parameters

lpszUrlName
Address of a string that contains the name of the source corresponding to the cache entry.

Return Values

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_FILE_NOT_FOUND	The file is not in the cache.
ERROR_ACCESS_DENIED	The file is in use.

FindCloseUrlCache

Closes the specified enumeration handle.

BOOL FindCloseUrlCache(
IN HANDLE *hEnumHandle* **);**

Parameters

hEnumHandle
Handle returned by a previous call to the FindFirstUrlCacheEntry function.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**.

See Also

FindFirstUrlCacheEntry

FindFirstUrlCacheEntry

Begins the enumeration of the cache.

HANDLE FindFirstUrlCacheEntry (
IN LPCSTR *lpszUrlSearchPattern*,
OUT LPINTERNET_CACHE_ENTRY_INFO *lpFirstCacheEntryInfo*,
IN OUT LPDWORD *lpdwFirstCacheEntryInfoBufferSize* **);**

Parameters

lpszUrlSearchPattern
Address of a string that contains the source name pattern to search for. If this parameter is NULL, the function uses *.*. (In this version we may implement only *.* semantics).

lpFirstCacheEntryInfo
Address of an INTERNET_CACHE_ENTRY_INFO structure.

lpdwFirstCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpFirstCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

Return Values

Returns a handle that the application can use in the FindNextUrlCacheEntry function to retrieve subsequent entries in the cache. If the function fails, the return value is NULL. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_INSUFFICIENT_BUFFER	The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information

Remarks

This function and the FindNextUrlCacheEntry function return variable size information. In order to not have the enumeration terminate due to **ERROR_INSUFFICIENT_BUFFER**, an application should create one buffer of the size specified by the MAX_CACHE_ENTRY_INFO_SIZE value, and pass the address of the buffer repeatedly to all the enumeration functions. After the function succeeds, another buffer may be used of the size returned by *lpdwCacheEntryInfoBufferSize* to keep the returned information. Be careful to fix up the pointer elements while copying the buffer.

See Also

FindNextUrlCacheEntry

FindNextUrlCacheEntry

Retrieves the next entry in the cache.

BOOL FindNextUrlCacheEntry(
IN HANDLE *hEnumHandle*,
OUT LPINTERNET_CACHE_ENTRY_INFO *lpNextCacheEntryInfo*,
IN OUT LPWORD *lpdwNextCacheEntryInfoBufferSize* **);**

Parameters

hEnumHandle
Enumeration handle obtained from a previous call to FindFirstUrlCacheEntry.

lpNextCacheEntryInfo
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about cache entry.

lpdwNextCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpNextCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

Return Values

Returns **TRUE** if successful or **FALSE** otherwise. To get extended error information, call **GetLastError**. Possible error values include:

Value	Meaning
ERROR_NO_MORE_FILES	The enumeration completed.
ERROR_INSUFFICIENT_BUFFER	The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information.

See Also

FindFirstUrlCacheEntry

Structures

This list identifies the WinInet data structures and their uses.

GOPHER_ATTRIBUTE_TYPE

Contains the relevant information of a single Gopher attribute for an object.

```
typedef struct {
    DWORD CategoryId
    DWORD AttributeId
    union {
        GOPHER_ADMIN_ATTRIBUTE Admin;
        GOPHER_MOD_DATE_ATTRIBUTE ModDate;
        GOPHER_SCORE_ATTRIBUTE Score;
        GOPHER_SCORE_RANGE_ATTRIBUTE ScoreRange;
        GOPHER_SITE_ATTRIBUTE Site;
        GOPHER_ORGANIZATION_ATTRIBUTE Organization;
        GOPHER_LOCATION_ATTRIBUTE Location;
        GOPHER_GEOGRAPHICAL_LOCATION_ATTRIBUTE GeographicalLocation;
        GOPHER_TIMEZONE_ATTRIBUTE TimeZone;
        GOPHER_PROVIDER_ATTRIBUTE Provider;
        GOPHER_VERSION_ATTRIBUTE Version;
        GOPHER_ABSTRACT_ATTRIBUTE Abstract;
        GOPHER_VIEW_ATTRIBUTE View;
        GOPHER_VERONICA_ATTRIBUTE Veronica;
        GOPHER_ASK_ATTRIBUTE_TYPE Ask;
        GOPHER_UNKNOWN_ATTRIBUTE Unknown;
    } AttributeType;
} GOPHER_ATTRIBUTE_TYPE, *LPGOPHER_ATTRIBUTE_TYPE;
```

Members

CategoryId

Gopher name for the attribute. The possible values include:

GOPHER_CATEGORY_ID_ALL

GOPHER_CATEGORY_ID_INFO

GOPHER_CATEGORY_ID_ADMIN

GOPHER_CATEGORY_ID_VIEWS

GOPHER_CATEGORY_ID_ABSTRACT

GOPHER_CATEGORY_ID_VERONICA

GOPHER_CATEGORY_ID_UNKNOWN

AttributeId

Identifier of the structure contained in the**AttributeType** member. The possible values include:

GOPHER_ATTRIBUTE_ID_ADMIN

GOPHER_ATTRIBUTE_ID_MOD_DATE

GOPHER_ATTRIBUTE_ID_TTL

GOPHER_ATTRIBUTE_ID_SCORE

GOPHER_ATTRIBUTE_ID_RANGE

GOPHER_ATTRIBUTE_ID_SITE

GOPHER_ATTRIBUTE_ID_ORG

GOPHER_ATTRIBUTE_ID_LOCATION

GOPHER_ATTRIBUTE_ID_GEOG
GOPHER_ATTRIBUTE_ID_TIMEZONE
GOPHER_ATTRIBUTE_ID_PROVIDER
GOPHER_ATTRIBUTE_ID_VERSION
GOPHER_ATTRIBUTE_ID_ABSTRACT
GOPHER_ATTRIBUTE_ID_VIEW
GOPHER_ATTRIBUTE_ID_TREEWALK
GOPHER_ATTRIBUTE_ID_UNKNOWN

AttributeType

Actual setting for the Gopher attribute. The specific value of **AttributeType** depends on the **AttributeId** member. The definitions of the various attribute structures is available in WININET.H.

See Also

GopherGetAttribute

GOPHER_FIND_DATA

Contain information retrieved by the GopherFindFirstFile and InternetFindNextFile functions.

```
typedef struct {
    TCHAR DisplayString[MAX_GOPHER_DISPLAY_TEXT + 1];
    DWORD GopherType;
    DWORD SizeLow;
    DWORD SizeHigh;
    FILETIME LastModificationTime;
    TCHAR Locator[MAX_GOPHER_LOCATOR_LENGTH + 1];
} GOPHER_FIND_DATA, FAR *LPGOPHER_FIND_DATA;
```

Members

DisplayString
String that contains the friendly name of an object. An application can display this string to allow the user to select the object.

GopherType
Mask of flags that describe the item returned.

FileSizeLow
Low 32 bits of the file size.

FileSizeHigh
High 32 bits of the file size.

LastModificationTime
Time when the file was last modified.

Locator
String that identifies the file. An application can pass the locator string to GopherOpenFile or GopherFindFirstFile.

See Also

GopherFindFirstFile

INTERNET_ASYNC_RESULT

Contains the result of a call to an asynchronous function. This structure is used with the InternetStatusCallback function.

```
typedef struct {
    DWORD dwResult;
    DWORD dwError;
} INTERNET_ASYNC_RESULT, * LPINTERNET_ASYNC_RESULT;
```

Members

dwResult
HINTERNET, DWORD or BOOL return code from an asynchronous function.

dwError
Error code if **dwResult** indicates that the function failed. If the operation succeeded, this member usually contains **ERROR_SUCCESS**.

See Also

InternetStatusCallback

INTERNET_CACHE_ENTRY_INFO

Contains information about an entry in the cache.

```
typedef struct _CACHE_ENTRY_INFO {
    DWORD dwStructSize;
    LPSTR lpszSourceUrlName;
    LPTSTR lpszLocalFileName;
    DWORD CacheEntryType;
    DWORD dwUseCount;
    DWORD dwHitRate;
    DWORD dwSizeLow;
    DWORD dwSizeHigh;
    FILETIME LastModifiedTime;
    FILETIME ExpireTime;
    FILETIME LastAccessTime;
    FILETIME LastSyncTime;
    LPBYTE lpHeaderInfo;
    DWORD dwHeaderInfoSize;
    LPTSTR lpszFileExtension;
    DWORD dwReserved;
} INTERNET_CACHE_ENTRY_INFO, *LPINTERNET_CACHE_ENTRY_INFO;
```

Members

dwStructSize
Size, in bytes, of this structure.

lpszSourceUrlName
Address of a string that contains the URL name. The string occupies memory area at the end of this structure.

lpszLocalFileName
Address of a string that contains the local filename. The string occupies memory area at the end of this structure.

CacheEntryType
Cache type bit mask. Can be one of these values:

Value	Meaning
NORMAL_CACHE_ENTRY	Normal cache entry, may be deleted to recover space for new entries.
STABLE_CACHE_ENTRY	sSable cache entry such as graphic and audio/video files, may be deleted to recover space for the new entries only when there is no more NORMAL_CACHE_ENTRY.
STICKY_CACHE_ENTRY	These entries will never be removed automatically by the cache management system.
SPARSE_CACHE_ENTRY	This cache entry is incomplete.
OCX_CACHE_ENTRY	Special OCX type cache entry.

dwUseCount
Current users count of the cache entry.

dwHitRate
Number of times the cache entry was retrieved

dwSizeLow
Low-order double word of the file size.

dwSizeHigh
High-order double word of the file size.

LastModifiedTime
Last modified time of this URL in GMT format.

ExpireTime
Expiration time of this file in GMT format.

LastAccessTime
Last accessed time in GMT format.

LastSyncTime

lpHeaderInfo
Address of a buffer that contains the header information. The buffer occupies memory at the end of this structure.

dwHeaderInfoSize
Size of the **lpHeaderInfo** buffer.

lpszFileExtension
Address of a string that contains the file extension used to retrieve the data as a file. The string occupies memory area at the end of this structure.

dwReserved
Reserved for future use.

Remarks

The **MAX_CACHE_ENTRY_INFO_SIZE** value defines the maximum size of the **INTERNET_CACHE_ENTRY_INFO** structure that could be returned by the implementation of the cache functions. Thus passing in a buffer of this size in functions returning **INTERNET_CACHE_ENTRY_INFO** guarantees that the function does not fail because of insufficient buffer.

INTERNET_CERTIFICATE_INFO

Contains certificate information returned from the server. This structure is used by the InternetQueryOption function.

```
typedef struct {
    FILETIME ftExpiry;
    FILETIME ftStart;
    LPTSTR lpszSubjectInfo;
    LPTSTR lpszIssuerInfo;
    LPTSTR lpszProtocolName;
    LPTSTR lpszSignatureAlgName;
    LPTSTR lpszEncryptionAlgName;
    DWORD dwKeySize;
} INTERNET_CERTIFICATE_INFO, * LPINTERNET_CERTIFICATE_INFO;
```

Members

ftExpiry
FILETIME structure that contains the date the certificate expires.

ftStart
FILETIME structure that contains the date the certificate becomes valid.

lpszSubjectInfo
Address of a buffer that contains the name of organization, site, and server for which the certificate was issued.

lpszIssuerInfo
Address of a buffer that contains the name of the orgainzation, site, and server that issued the certificate.

lpszProtocolName
Address of a buffer that contains the name of the protocol used to provide the secure connection.

lpszSignatureAlgName
Address of a buffer that contains the name of the algorithm used for signing the certificate.

lpszEncryptionAlgName
Address of a buffer that contains the name of the algorithm used for doing encryption over the secure channel (SSL/PCT) connection.

dwKeySize
Size, in bytes, of the key.

See Also

InternetQueryOption

INTERNET_PREFETCH_STATUS

Contains the status of a prefetch download operation.

```
typedef struct {
    DWORD dwStatus;
    DWORD dwSize;
} INTERNET_PREFETCH_STATUS, * LPINTERNET_PREFETCH_STATUS;
```

Members

dwStatus
Status of the download. Can be one of these values:

INTERNET_PREFETCH_PROGRESS	The operation is in progress.
INTERNET_PREFETCH_COMPLETE	The operation has completed.
INTERNET_PREFETCH_ABORTED	The operation was aborted.

dwSize
Size, in bytes, of data downloaded so far.

INTERNET_PROXY_INFO

Contains information that is supplied with INTERNET_OPTION_PROXY to get or set proxy information on a handle obtained from a call to the InternetOpen function.

```
typedef struct {
    DWORD dwAccessType;
    LPCTSTR lpszProxy;
    LPCTSTR lpszProxyBypass;
} INTERNET_PROXY_INFO, * LPINTERNET_PROXY_INFO;
```

Members

dwAccessType
Access type. Can be one of these values:

Value	Meaning
INTERNET_OPEN_TYPE_DIRECT	
INTERNET_OPEN_TYPE_PROXY	
INTERNET_OPEN_TYPE_PRECONFIG	Applies only when setting proxy information.

lpszProxy
Proxy server list.

lpszProxyBypass
Proxy bypass list.

INTERNET_SCHEME

Defines the flags used with the nScheme member of URL_COMPONENTS structure.

```
typedef enum {
    INTERNET_SCHEME_PARTIAL = -2,
    INTERNET_SCHEME_UNKNOWN = -1,
    INTERNET_SCHEME_DEFAULT = 0,
    INTERNET_SCHEME_FTP,
    INTERNET_SCHEME_GOPHER,
    INTERNET_SCHEME_HTTP,
    INTERNET_SCHEME_HTTPS,
    INTERNET_SCHEME_FILE,
    INTERNET_SCHEME_NEWS,
    INTERNET_SCHEME_MAILTO,
    INTERNET_SCHEME_FIRST = INTERNET_SCHEME_FTP,
    INTERNET_SCHEME_LAST = INTERNET_SCHEME_MAILTO
} INTERNET_SCHEME, * LPINTERNET_SCHEME;
```

INTERNET_VERSION_INFO

Contains the version number of the DLL that contains the Windows Internet functions (WININET.DLL). This structure is used when passing the INTERNET_OPTION_VERSION flag to the InternetQueryOption function.

```
typedef struct {
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
} INTERNET_VERSION_INFO, * LPINTERNET_VERSION_INFO;
```

Members

dwMajorVersion
Major version number.

dwMinorVersion
Minor version number.

URL_COMPONENTS

Contains the constituent parts of a URL. This structure is used with the InternetCrackUrl and InternetCreateUrl functions.

```
typedef struct {
    DWORD dwStructSize;
    LPSTR lpszScheme;
    DWORD dwSchemeLength;
    INTERNET_SCHEME nScheme;
    LPSTR lpszHostName;
    DWORD dwHostNameLength;
    INTERNET_PORT nPort;
    LPSTR lpszUserName;
    DWORD dwUserNameLength;
    LPSTR lpszPassword;
    DWORD dwPasswordLength'
    LPSTR lpszUrlPath;
    DWORD dwUrlPathLength;
    LPTSTR lpszExtraInfo;
    DWORD dwExtraInfoLength;
} URL_COMPONENTS;
```

Members

dwStructSize
Size, in bytes, of this structure. Used for version checking.

lpszScheme
Address of a buffer that contains the scheme name.

dwSchemeLength
Length of the scheme name.

nScheme
Enumerated scheme type (if known). For a list of scheme types, see INTERNET_SCHEME.

lpszHostName
Address of a buffer that contains the host name.

dwHostNameLength
Length of the host name.

nPort
Converted port number.

lpszUserName
Address of a buffer that contains the user name.

dwUserNameLength
Length of the user name.

lpszPassword
Address of a buffer that contains the password.

dwPasswordLength
Length of the password.

lpszUrlPath
Address of a buffer that contains the URL path.

dwUrlPathLength

Length of the URL path.

lpszExtraInfo

Address of a buffer that contains the extra information (for example, ?foo or #foo).

dwExtraInfoLength

Length of the extra information.

Remarks

For InternetCrackUrl, if a pointer member and its corresponding length member are both zero, that component is not returned. If the pointer member is NULL but the length member is not zero, both the pointer and length members are returned. If both pointer and corresponding length members are non-zero, the pointer member points to a buffer where the component is copied. The component may be un-escaped, depending on the *dwFlags* parameter of InternetCrackUrl.

For InternetCreateUrl, the pointer members should be NULL if the component is not required. If the corresponding length member is zero, the pointer member is the address of a zero-terminated string. If the length member is not zero, it is the string length of the corresponding pointer member

Error Codes

The HTTP functions control the transmission and content of HTTP requests.

The Win32 Internet functions return Win32 error codes where appropriate. The following error codes are specific to the Win32 Internet functions:

ERROR_INTERNET_OUT_OF_HANDLES

No more handles could be generated at this time.

ERROR_INTERNET_TIMEOUT

The request has timed out.

ERROR_INTERNET_EXTENDED_ERROR

An extended error was returned from the server. This is typically a string or buffer containing a verbose error message. Call InternetGetLastResponseInfo to retrieve the error text.

ERROR_INTERNET_INTERNAL_ERROR

An internal error has occurred.

ERROR_INTERNET_INVALID_URL

The URL is invalid.

ERROR_INTERNET_UNRECOGNIZED_SCHEME

The URL scheme could not be recognized, or is not supported.

ERROR_INTERNET_NAME_NOT_RESOLVED

The server name could not be resolved.

ERROR_INTERNET_PROTOCOL_NOT_FOUND

The requested protocol could not be located.

ERROR_INTERNET_INVALID_OPTION

A request to InternetQueryOption or InternetSetOption specified an invalid option value.

ERROR_INTERNET_BAD_OPTION_LENGTH

The length of an option supplied to InternetQueryOption or InternetSetOption is incorrect for the type of option specified.

ERROR_INTERNET_OPTION_NOT_SETTABLE

The request option can not be set, only queried.

ERROR_INTERNET_SHUTDOWN

The Win32 Internet support is being shutdown or unloaded.

ERROR_INTERNET_INCORRECT_USER_NAME

The request to connect and login to an FTP server could not be completed because the supplied user name is incorrect.

ERROR_INTERNET_INCORRECT_PASSWORD

The request to connect and login to an FTP server could not be completed because the supplied password is incorrect.

ERROR_INTERNET_LOGIN_FAILURE

The request to connect to and login to an FTP server failed.

ERROR_INTERNET_INVALID_OPERATION

The requested operation is invalid.

ERROR_INTERNET_OPERATION_CANCELLED

The operation was canceled, usually because the handle on which the request was operating was closed before the operation completed.

ERROR_INTERNET_INCORRECT_HANDLE_TYPE

The type of handle supplied is incorrect for this operation.

ERROR_INTERNET_INCORRECT_HANDLE_STATE

The requested operation can not be carried out because the handle supplied is not in the correct state.

ERROR_INTERNET_NOT_PROXY_REQUEST

The request can not be made via a proxy.

ERROR_INTERNET_REGISTRY_VALUE_NOT_FOUND

A required registry value could not be located.

ERROR_INTERNET_BAD_REGISTRY_PARAMETER

A required registry value was located but is an incorrect type or has an invalid value.

ERROR_INTERNET_NO_DIRECT_ACCESS

Direct network access cannot be made at this time.

ERROR_INTERNET_NO_CONTEXT

An asynchronous request could not be made because a zero context value was supplied.

ERROR_INTERNET_NO_CALLBACK

An asynchronous request could not be made because a callback function has not been set.

ERROR_INTERNET_REQUEST_PENDING

The required operation could not be completed because one or more requests are pending.

ERROR_INTERNET_INCORRECT_FORMAT

The format of the request is invalid.

ERROR_INTERNET_ITEM_NOT_FOUND

The requested item could not be located.

ERROR_INTERNET_CANNOT_CONNECT

The attempt to connect to the server failed.

ERROR_INTERNET_CONNECTION_ABORTED

The connection with the server has been terminated.

ERROR_INTERNET_CONNECTION_RESET

The connection with the server has been reset.

ERROR_INTERNET_FORCE_RETRY

ERROR_INTERNET_ZONE_CROSSING

ERROR_INTERNET_MIXED_SECURITY

ERROR_INTERNET_SSL_CERT_CN_INVALID

ERROR_INTERNET_HANDLE_EXISTS

The request failed because the handle already exists.

ERROR_FTP_TRANSFER_IN_PROGRESS

The requested operation can not be made on the FTP session handle because an operation is already in progress

ERROR_FTP_DROPPED

The FTP operation was not completed because the session was aborted.

ERROR_GOPHER_PROTOCOL_ERROR

An error was detected whilst parsing data returned from the gopher server.

ERROR_GOPHER_NOT_FILE

The request must be made for a file locator.

ERROR_GOPHER_DATA_ERROR

An error was detected while receiving data from the gopher server.

ERROR_GOPHER_END_OF_DATA

The end of the data has been reached.

ERROR_GOPHER_INVALID_LOCATOR

The supplied locator is not valid.

ERROR_GOPHER_INCORRECT_LOCATOR_TYPE

The type of the locator is not correct for this operation.

ERROR_GOPHER_NOT_GOPHER_PLUS

The requested operation can only be made against a Gopher+ server, or with a locator that specifies a Gopher+ operation.

ERROR_GOPHER_ATTRIBUTE_NOT_FOUND

The requested attribute could not be located.

ERROR_GOPHER_UNKNOWN_LOCATOR

The locator type is unknown.

ERROR_HTTP_HEADER_NOT_FOUND

The requested header could not be located.

ERROR_HTTP_DOWNLEVEL_SERVER

The server did not return any headers.

ERROR_HTTP_INVALID_SERVER_RESPONSE

The server response could not be parsed.

ERROR_HTTP_INVALID_HEADER

The supplied header is invalid.

ERROR_HTTP_INVALID_QUERY_REQUEST

The request made to HttpQueryInfo is invalid

ERROR_HTTP_HEADER_ALREADY_EXISTS

The header could not be added because it already exists.

Executive Summary

This document provides a description of the mechanism for downloading and installing code for COM Objects (components) using the Microsoft Active Internet Platform (Sweeper). This mechanism is used internally by the Microsoft Internet Explorer for downloading OLE Controls inserted in HTML pages.

- Internet Component Download is a mechanism for downloading and installing code for COM objects(1).
- Details are presented for how OLE Control authors should package their objects to be downloaded and installed automatically.
- A new system API CoGetClassObjectFromURL is presented for downloading COM components. Other "safe code-download" needs can be met using the lower-level **WinVerifyTrust** service, or possibly using a high-level "Setup" OLE Control. Future releases may expose a more sophisticated component download interface.
- Internet Component Download makes use of an Internet Search Path to search various "Object Stores" for download-able code.
- Internet Component Download installs code in a permanent store. This document details a migration strategy for future releases to convert this store into a cache that discards unpopular components.

Introduction

Internet Component Download is a system service for downloading, certificate checking, and installing COM component code from the Internet. This service is used by applications (e.g. web browsers) to automatically download and install COM Objects from code repositories on the Internet. This document explains how code authors should prepare their components for automatic download. It then describes the interface for the component download mechanism, and finally provides some additional implementation details, including a description of the Internet Search Path service which allows searching for download-able code from a series of "Object Stores".

Component Download is used within Microsoft Internet Explorer in order to automatically download OLE Controls inserted inside HTML pages using the <OBJECT> element in HTML(2). The mechanism for downloading components is exposed in an API that may be used in various other OLE containers. OLE Control developers should follow the guidelines outlined below to package their controls so that they can be downloaded automatically by any container that uses the Component Download mechanism.

Packaging component code for automatic download

ISVs and authors of COM Objects for the internet should package their implementations so that they may be downloaded automatically by web browsers such as the Microsoft Internet Explorer. Such objects will be downloaded, for instance, when parsing the OBJECT tag in HTML(3). For details, see the OBJECT tag specification.

Interpreting the "CODEBASE" URL

The "CODEBASE" attribute in an OBJECT tag contains a URL pointing to the implementation of a given COM object. This URL is of critical importance for component download, because it must specify all files necessary to implement a particular COM object. HTML authors can author "CODEBASE" URL to point to one of three file types. Component developers should choose one of the packaging schemes below for their COM Objects:

1. A single **PE** (portable executable, e.g. an .OCX, a .DLL, or a .EXE): This single executable is downloaded, installed, and registered in one fell swoop. This is the simplest way to package a single-file OLE control, but (a) it will not use file compression, (b) it will not be platform independent except with HTTP.
2. A **.CAB** (cabinet) file: This file contains one or more files, all of which are downloaded together in a compressed cabinet.<Footnote Code Text> Care must be taken so that the cabinet file contains only those files that must *necessarily* be downloaded (e.g. the OCX executable itself). Any additional helper DLLs (e.g. MFC) may have already been installed and if so should not be bundled into the cabinet(4). Exactly one file in the cabinet is an .INF file providing further installation information. This .INF file may refer to files in the .CAB as well to files at other URLs. This mechanism requires authoring of a .INF and packaging of a .CAB file, but in return it provides file compression. It will not be platform independent, however, except with HTTP format negotiation.
3. A stand-alone **.INF** file: This file specifies various files that need to be downloaded and setup for the OCX to run. The syntax of the .INF file allows (a) URLs pointing to files to download, and (b) platform independence (by enumerating files for various platforms). This mechanism provides platform independence for non-HTTP servers.

Registry settings and self-extracting .exes

It's recommended to use self-registering code for Internet Component Download, because the .INF format used by Internet Component Download (see below) does *not* provide syntax for changing registry information (for security reasons). For .OCXs, .DLLs, and .EXEs marked as "OleSelfRegister" in the Version resource, Internet Component Download will try to run self-registration. For .DLLs, this means loading the .DLL library and calling the DllRegisterServer entry point, if available. For .EXEs, this means *running* the .EXE with the run-time parameter of "/RegServer". This ensures that COM Objects implemented as local servers (e.g. winword.exe) are registered correctly. If an object is not marked as "OleSelfRegister" but registration is necessary, or if it is desired to over-ride the "OleSelfRegister" flag, one can add the following to an .INF file (see .INF setup-script format below):

```
[foo.ocx]
RegisterServer=no ; don't register even if marked OleSelfRegister
  or
RegsiterServer=yes ; register this even if not marked OleSelfRegister. This
is the typical workaround for getting old
      ; controls to register
```

Code that is downloaded via Internet Component Download may be a self-extracting .EXE because Internet Component Download ignores the "OleSelfRegister" flag if the main URL for code download points directly at a .EXE file. In this case it is assumed that this is a self-registering .EXE, and this enables self-extracting .EXEs to work correctly as long as they ignore the "/regsvr" command-line parameter. Supporting self-extracting .EXEs enables very complex setup mechanisms to be launched automatically via Internet Component Download. However, if a self-extracting .EXE is called via this mechanism, then any components that it installs will not be automatically tracked by Internet Component Download (see Appendix on ModuleUsage section in registry). Such components are permanently installed and are not marked by Internet Component Download for future cleanup.

Including version number in the "CODEBASE" URL

Besides the actual address of code, the "CODEBASE" URL may also include an optional version number using the following syntax: "CODEBASE=http://www.foo.com/bar.ocx#Version=a,b,c,d". The Internet Component Download mechanism will download and install the file only if the specified version number is *more recent* than any existing version of the same file currently installed in the system. (see Appendix on registry details for more information). If a version number is not specified with a file, it is assumed that any version installed on the system is recent enough(5).

If the version number specified in the CODEBASE attribute is "-1,-1,-1,-1", then Internet Component Download will *always* try to download the *latest* version of the desired component. Note that this can be a costly effort involving many network transactions, especially if the Internet Search Path is searched for newest versions of an object (see below for details on Internet Search Path). Note also that because of the Internet Search Path, it is possible for the Component Download service to try to download code in the absence of a CODEBASE attribute. In fact, if the CODEBASE attribute is the URL fragment "#Version=-1,-1,-1,-1", then there is no specific location to download code from, but the Internet Search Path will still be searched to find the *latest* version of an object(6).

Platform independence and HTTP

When code to downloaded is on an HTTP server, the HTTP Accept header MIME request type may be used to specify which platform the code is to run on, thus allowing platform independence of the "CODEBASE" URL.

The following MIME types will be used to describe PEs (portable executables - .EXE, .DLL, .OCX), cabinet files (.CAB), and setup scripts (.INF):(7)

File description	MIME Type
PE (portable executable) - .EXE, .DLL, .OCX	application/x-pe-%opersys%-%cpu%
Cabinet files - .CAB	application/x-cabinet-%opersys%-%cpu%
Setup scripts - .INF (platform independent)	application/x-setupscript

%opersys% and %cpu% above will specify the operating system and CPU for the desired platform downloaded components will be executed on. For example, the MIME type for a Win32 cabinet file running on an Intel® x86-architecture processor would be application/x-cabinet-win32-x86.

The following are valid values for %opersys% and %cpu%:

Valid values for %opersys%	Meaning
win32	32-bit Windows® operating systems (Windows95 or Windows NT)
mac	Macintosh® operating system
<other>	will be defined as necessary

Valid values for %cpu%	Meaning
x86	Intel® x86 family of processors
ppc	Motorola® PowerPC architecture
mips	MIPS® architecture processors
alpha	DEC® Alpha architecture

When the code is on a non-HTTP server (e.g. at a local LAN location), a .INF file can be used to achieve platform independence by specifying different URLs for files to be downloaded for different platforms. (see the section below on platform independence in .INF files)

.CAB format

The .CAB format used for Internet Component Download is a non-proprietary format based on Lempel-Ziv compression. The Microsoft Internet SDK includes a free tool called "diantz.exe" that will package cabinet files into this non-proprietary format. There no specification of this .CAB format publicly available, although such a specification will be distributed as soon as possible.

Use of the DIANTZ.EXE tool for creating .CAB cabinet files

The DIANTZ.EXE tool takes a .DDF "directive file" specifying which files to combine into a cabinet. The syntax for using this tool from a command line is:

```
DIANTZ.EXE /f <directive file.ddf>
```

The example directive file below, CIRC3Z.DDF, would be used for creating a cabinet file containing two files - circ3.inf and circ3.ocx. It should be straightforward to add to this list of files...

```
; DIAMOND directive file for CIRC3.OCX+CIRC3.INF
.OPTION EXPLICIT                        ; Generate errors on variable
typos
.Set CabinetNameTemplate=CIRC3Z.CAB
;** The files specified below are stored, compressed, in cabinet files
.Set Cabinet=on
.Set Compress=on
circ3.INF
circ3.OCX
```

Note It is possible to use the "code-signing" utilities to sign entire cabinet files using a digital certificate. However, in order to do this, it is necessary to add the following linesto the .DDF file before the list of files for inclusion in the cabinet.

```
;Reserve space for PKS#7 Code Signature
.Set ReservePerCabinetSize=2048
```

If a cabinet file is signed, it is assumed that *every* file inside the cabinet is trusted, including .INF and .INI files. This has two advantages:

1 It is now possible to include powerful .INFs inside a trusted cabinet

2 By signing an entire cabinet the time for verifying digital certificates can be sped up due to the cabinet compression

.INF setup-script format

Here is a sample .INF file that demonstrates the syntax that is understood by the Component Download service. **Note: Only the .INF syntax below may be used to write setup scripts for Internet Component Download. Due to security reasons the system standard (SetupX) .INF setup is not called to install components with setup scripts, and instead the limited INF syntax below is the only legal format for Internet Component Download.** See Future Directions below for plans for eventually supporting other .INF formats.

```
;Sample INF file for CIRC3.OCX
[Add.Code]
circ3.ocx=circ3.ocx
random.dll=random.dll
mfc40.dll=mfc40.dll
foo.ocx=foo.ocx

[circ3.ocx]
; lines below specify that the specified circ3.ocx (clsid, version) needs
to be installed on
; the system. If doesn't exist already, can be downloaded from the given
location (a .CAB)
; note: if "thiscab" is specified instead of the file location, it is
assumed that the
; desired file is present in the same .CAB cabinet that the INF originated
from
; otherwise, if the location pointed to is a different .CAB, the new
cabinet is also downloaded and
; unpacked in order to extract the desired file
file=http://www.code.com/circ3/circ3.cab
clsid={9DBAFCCF-592F-101B-85CE-00608CEC297B}
FileVersion=1,0,0,143

[random.dll]
; lines below specify that the random.dll needs to be installed in the
system
; if this doesn't exist already, it can be downloaded from the given
location.
file=http:// www.code.com/circ3/random.dll
; Note that the FileVersion is option, and it may also be left empty,
meaning that any version is ok.
FileVersion=
DestDir=10


; DestDir can be set to 10 or 11 ( LDID_WIN or LDID_SYS by INF convention)
; this places files in \windows or \windows\system, respectively
; if no dest dir specified (typical case), code is installed in the fixed
occache directory.

[mfc40.dll]
; leaving the file location empty specifies that the installation
; needs mfc40 (version 4,0,0,5), but it should not be downloaded.
```

```
; if this file is not already present on the client machine, component
download fails
file=
FileVersion=4,0,0,5

[foo.ocx]
; leaving the file location empty specifies that the installation
; needs the specified foo.ocx (clsid, version), but it should not be
downloaded.
; if this file is not already present on the client machine, component
download fails
file=
clsid={DEADBEEF-592F-101B-85CE-00608CEC297B}
FileVersion=1,0,0,143
```

Platform independence in .INF files

It is possible to create platform-independent setup scripts that pull files from different locations depending on the desired platform. Internet Component Download .INF files will use a scheme similar to the one described above under "Platform Independence and HTTP". Specifically, a sample platform-independent .INF file would include a text such as the following:

```
[circ3.ocx]
; lines below specify that the specified circ3.ocx (clsid, version) needs
to be installed on
; the system. If doesn't exist already, can be downloaded from the given
location (a .CAB)
file-win32-x86=file://products/release/circ3/x86/circ3.cab
file-win32-mips=file://products/release/circ3/mips/circ3.cab
file-mac-ppc=ignore
  ; the 'ignore' keyword means that this file is not needed for this
platform

clsid={9DBAFCCF-592F-101B-85CE-00608CEC297B}
FileVersion=1,0,0,143
```

Thus the "file=" syntax used in the .INF file is expanded to "file-%opersys%-%cpu =", allowing the .INF file to specify multiple locations where various platform-dependent modules can be found and downloaded. See the section above for valid values for %opersys% and %cpu%.

The Internet Component Download interface

The Internet Component Download service is exposed via a single function, CoGetClassObjectFromURL(). This system function is called by an application that wishes to download, verify, and install code for an OLE component. The function is used in the implementation of Microsoft® Internet Explorer. The implementation uses **URL Moniker** to asynchronously download code, and it uses the **WinVerifyTrust** service to verify validity of the code.

Related Documents	**Filename**
URL Moniker specification	urlmon.doc
Asynchronous Moniker specification	asyncmon.doc
WinVerifyTrust specification	wintrust.doc

Architecture

The diagram below shows the implementation architecture for the Internet Component Download mechanism and its relation to other system services:

{ewc msdncd, EWGraphic, grpsweeper 20 /a "sweeper.bmp"}

Technical Details

This section describes technical details of the Internet Component Download API used by applications (e.g. web browsers) to download and install COM Object code.

CoGetClassObjectFromURL

This function will return a factory object for a given rclsid. If no CLSID is specified (CLSID_NULL), this function will choose the appropriate CLSID for interpreting the Internet MIME type specified in szContentType. If the desired object is installed on the system, it is instantiated. Otherwise, the necessary code is downloaded and installed from the location specified in szCodeURL or from an Object Store on the Internet Search Path (see below).

STDAPI CoGetClassObjectFromURL (
[in] REFCLSID *rclsid*,
[in] LPCWSTR *szCodeURL*,
[in] DWORD *dwFileVersionMS*,
[in] DWORD *dwFileVersionLS*,
[in] LPCWSTR *szContentTYPE*,
[in] LPBINDCTX *pBindCtx*,
[in] DWORD *dwClsContext*,
[in] LPVOID *pvReserved*,
[in] REFIID *riid*,
[out] VOID ***ppv* **);**

This "download and install" process involves the following steps:

1 Downloading the necessary file(s) (.CAB, .INF, or executable) using URL Moniker(s).
2 Call WinVerifyTrust to ensure that all downloaded files are safe to install.
3 Complete self-registration of all COM components(8)
4 Add registry entries to keep track of downloaded code (see Appendix on Registry Details)
5 Call CoGetClassObject for the desired rclsid.

CoGetClassObjectFromURL accepts the following arguments:

REFCLSID*rclsid*	CLSID of the COM object that needs to be installed. If value is CLSID_NULL, then szContentType is used to determine the CLSID.
LPCWSTR*szCodeURL*	URL pointing to the code for the COM object. This may point to an executable, to an .INF file, or to a .CAB file (see below for details). If this value is NULL, then Internet Component Download will still attempt to download the desired code from an Object Store on the Internet Search Path.
DWORD*dwFileVersionMS*	Major version number for the object that needs to be installed. If this value and the next are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired, which means that Internet Component Download will *always* attempt to download new code.
DWORD*dwFileVersionLS*	Minor version number for the object that needs to be installed. If this value and the previous one are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired, which means that Internet Component Download will *always* attempt to download new code.
LPCWSTR*szContentType*	MIME type that needs to be understood by the installed COM object. If rclsid is CLSID_NULL, this string is used to determine the CLSID of the object that must be installed. Note: this parameter is only useful when trying to download a viewer for a particular media type, if the MIME type of the media is known but the CLSID is not.

LPBINDCTX*pBindCtx*	A bind context to use for downloading/installing component code. The client should register its IBindStatusCallback in this bind context to receive callbacks during the download and installation process. (See **Asynchronous Monikers** specification for details)
DWORD*dwClsContext*	Values taken from the CLSCTX enumeration specifying the execution context for the class object.
LPVOID*pvReserved*	Reserved value, must be set to NULL.
REFIID*riid*	The interface to obtain on the factory object (typically IClassFactory).
VOID ***ppv*	Pointer in which to store the interface pointer upon return if the call is synchronous.
S_OK	Success. ppv contains the requested interface pointer.
MK_S_ASYNCHRONOUS	Component code will be downloaded and installed asynchronously. The client will receive notifications through the IBIndStatusCallback interface it has registered on pBindCtx..
E_NOINTERFACE	The desired interface pointer is not available. Other CoGetClassObject error return values are also possible here..

In the common web-browser scenario, the values for parameters passed to this function are read directly from an HTML OBJECT tag. For example, the szCodeURL, dwFileVersionMS, and dwFileVersionLS are specified inside an <OBJECT> tag as "CODEBASE=http://www.foo.com/bar.ocx#Version=a,b,c,d", where szCodeURL is "http://www.foo.com/bar.ocx", dwFileVersionMS is MAKELONG(b, a), and dwFileVersionLS is MAKELONG(d, c).

Because the downloading and installation of code occurs asynchronously, CoGetClassObjectFromURL will often return immediately with a return value of E_PENDING. At this point, the IBindStatusCallBack mechanism is used to communicate the status of the download operation to the client(9). To participate in this communication, the client *must* implement IBindStatusCallback and register this interface in the pBindCtx passed into CoGetClassObjectFromURL using RegisterBindStatusCallback. The client can expect to be called with callback notifications for OnStartBinding (providing an IBinding for controlling the download), OnProgress (reporting progress), OnObjectAvailable (which returns the desired object interface pointer), and OnStopBinding (which returns error codes in case of an error). For further negotiations, the client *must* also implement ICodeInstall as described below.

Note The initial (beta) implementation of CoGetClassObjectFromURL will not handle system-wide simultaneous downloads of the *same* code. Similarly, it will not handle cases where different simultaneous downloads refer to the same piece of dependent code.

IBindStatusCallback::OnProgress

The client of CoGetClassObjectFromURL will receive notification about the download / install process via the provided IBindStatusCallback interface. During the download process , the following additional values (from the BINDSTATUS enumeration) may be passed back as the ulStatusCode parameter for IBindStatusCallback::OnProgress.

Parameters

BINDSTATUS_BEGINDOWNLOADCOMPONENTS
: The download operation has begun downloading code for COM components that will be installed before the object may be instantiated. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the component being downloaded.

BINDSTATUS_INSTALLINGCOMPONENTS
: The download operation has downloaded code and is installing it. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the component being installed.

BINDSTATUS_ENDDOWNLOADCOMPONENTS
: The download operation has finished downloading and installing all necessary code. The szStatusText accompanying OnProgress() provides the display name of the newly installed component.

ICodeInstall

A code install operation requires additional services from the client in order to complete the negotiation necessary for a download operation. Such services are requested using IBindStatusCallback::QueryInterface. The specific interface requested in IBindStatusCallback::QueryInterface is ICodeInstall. This interface must be implemented by a client of Internet Component Download.

interface ICodeInstallIParseDisplayNameInterfaces : IUnknown{
HRESULT GetWindow(
[out] HWND* *phwnd*
);
HRESULT OnCodeInstallProblem(
[in] ULONG *ulStatusCode*,
[in] *LPCWSTRszDestination*,
[in] LPCWSTR *szSource*,
[in] DWORD *dwReserved*
);
};

ICodeInstall::GetWindow

This function is called when Component Download needs to display user interface for verification of downloaded code(10). When a client is called with this function, it has the opportunity to clear the message queue of it's parent window before allowing UI to be displayed. If the client does not wish to display UI, code verification may continue, but components may fail to be installed.

Note ICodeInstall actually inherits from the IWindow interface (see documentation for URL Monikers), which consists of the single member function GetWindow. However, as far as any client code is concerned, the interface definition in this specification is still accurate.

Parameters

HWND **phwnd*

Client-provided HWND of the parent window for displaying code verification UI. If this parameter is NULL, the desktop window is used. If the value is INVALID_HANDLE_VALUE, or if the return value is S_FALSE, then no code verification UI will be displayed, and certain necessary components may not be installed.

Return Values

S_OK	Success.
S_FALSE	No window is available.
E_INVALIDARG	The argument is invalid.

ICodeInstall::OnCodeInstallProblem

This function is called when there is a problem with code installation. This notification gives the client a chance to resolve the problem, often by displaying UI, or by aborting the code installation process.

Note If the client does not understand the problem, it should return E_ABORT by default to abort the code installation process, because returning S_OK would imply retrying the operation.

Parameters

ULONG*ulStatusCode*
Status code describing what problem occurred. A member of CIP_STATUS.

LPCWSTR*szDestination*
The name of the existing file that was causing a problem. This may be the name of an existing file that needs to be overwritten, the name of a directory causing access problems, or the name of a drive that is full.

LPCWSTR*szSource*
Name of the new file to replace the existing file (if applicable).

DWORD*dwReserved*
Reserved for future use.

Return Values

S_OK	Continue the installation process. If there was an "access denied" or disk-full problem, retry the installation. If there was an existing file (newer *or* older version), overwrite it.
S_FALSE	Skip this particular file, but continue with the rest of the code installation process. Note: this is the typical response for the CIP_NEWER_VERSION_EXISTS case.
E_ABORT	Abort the code installation process.
E_INVALIDARG	The given arguments are invalid.

Remarks

The ulStatusCode parameter above is one of the following values:

```
typedef enum {
  CIP_DISK_FULL,
  CIP_ACCESS_DENIED,
  CIP_OLDER_VERSION_EXISTS,
  CIP_NEWER_VERSION_EXISTS,
  CIP_NAME_CONFLICT,
  CIP_TRUST_VERIFICATION_COMPONENT_MISSING
} CIP_STATUS;
```

CIP_DRIVE_FULL	The drive specified in szDestination is full..
CIP_ACCESS_DENIED	Access to the file specified in szDestination is denied.
CIP_OLDER_VERSION_EXISTS	An existing file (older version) specified in szDestination needs to be overwritten by the file specified in szSource.
CIP_NEWER_VERSION_EXISTS	A file exists (specified in szDestination) that is a newer version of a file to be installed (specified in szSource)

CIP_NAME_CONFLICT	A file exists (specified in szDestination) that has a naming conflict with a file to be installed (specified in szSource). The existing file is neither a newer nor an older version of the new filethey are mismatched but have the same file name.
CIP_TRUST_VERIFICATION_COMPONENT_MISSING	The code installation process cannot find the necessary component (WinVerifyTrust) for verifying trust in downloaded code. szSource specifies the name of the file that cannot be certified. The client should display UI asking the user whether or not to install the untrusted code, and should then return E_ABORT to abort the download, S_OK to continue anyway, or S_FALSE to skip this file but continue (usually dangerous).

Storing / Caching Downloaded Code

Code Download installs most new code in a permanent store in windows\occache(11). Some components (helper DLLs that need to be on the system PATH but currently are not) will also be installed in \windows and \windows\system. All downloaded code is registered using a new registry "Module Usage" section that keeps track of such code. Downloaded code is *not* removed automatically, but it is possible in the future to add UI to the Control Panel (or elsewhere) allowing a user to clean up this directory.

For future releases, it is also possible to convert this "permanent store" into a code cache that retains only popular downloaded code and deletes old unused code automatically. This migration plan justifies use of a permanent store for the first version. See Appendix for registry details on how downloaded code is listed in the registry and how a code cache could function in future releases.

Internet Search Path

When Internet Component Download is called to download code, it traverses the *Internet Search Path* to look for the desired component. This path is a list of *Object Store* servers that will be queried every time components are downloaded using CoGetClassObjectFromURL. This way, even if an <OBJECT> tag in an HTML document does not specify a CODEBASE location to download code for an embedded OLE Control, the Internet Component Download will still use the Internet Search Path to find the necessary code.

Internet Search Path syntax

The search path is specified in a string in the registry, under the key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Internet Settings\CodeBaseSearchPath. The value for this key is a string in the following format:

```
CodeBaseSearchPath = URL1; URL2; ... URLm; CODEBASE; URLm+1; ... URLn-1; URLn
```

Where each of *URL1* through *URLn* are absolute URLs pointing to HTTP servers acting as "Object Stores". When processing a call to CoGetClassObjectFromURL, the Internet Component Download service will *first* try downloading the desired code from the locations *URL1* through *URLm*, it will *then* try the location specified in the szCodeURL parameter (corresponding to the CODEBASE attribute in the <OBJECT> tag), and will *finally* try the locations specified in locations *URLm+1* through *URLn*.

Note That if the CODEBASE keyword is not included in the CodeBaseSearchPath key, then calls to CoGetClassObjectFromURL will never check the szCodeURL location for downloading code. By removing the CODEBASE keyword from the CodeBaseSearchPath, corporate intranet administrators can effectively disable Internet Component Download for corporate users.

Object Stores

An *Object Store* on the Internet Search Path is an HTTP server that services requests for downloadable code(12). During a call to CoGetClassObjectFromURL, Internet Component Download will try to download code from the various Object Stores on the search path. Specifically, an Object Store will receive an HTTP POST request with data in the format below(13).

```
CLSID={class id}
Version=a,b,c,d
MIMETYPE=<e>mimetype
```

All the values above are optional, although one of CLSID or MIMETYPE *must* be present. The Object Store should parse this information, check an internal database, and either fail the call, or **redirect** the HTTP request to the download-able code Cabinet file (.CAB), setup script (.INF), or portable executable (.EXE/.DLL/.OCX).

The POST parameters should be processed by the Object Store as follows:

- If CLSID is provided with no version number, then the most recent object matching the CLSID will be returned. If the CLSID is provided with Version, then the object matching the CLSID and with the *largest version number greater than or equal to* Version will be provided. If no object is available that matches the CLSID with a large enough version number, then the 404 error will be returned. MIMETYPE will be ignored when CLSID is provided.
- If no CLSID is provided, but MIMETYPE is provided, then the first object found in the database that matches the MIMETYPE will be returned. Version, if provided, is treated *exactly* as above. If neither CLSID or MIMETYPE is provided then the error return code 400 Bad Request will be returned.

In addition to the POST data described above, queries to Object Stores will also include HTTP headers for Accept (MIME type) and Accept-Language, thus specifying the desired platforms (see above for Platform Independence and HTTP) and language-localized implementation for a component. Note that these HTTP headers are added to all HTTP requests made by Internet Component Download. This allows Object Stores to serve different code implementations for differing platforms or even different languages.

Note Internet Component Download will use the *first successful response* from a server on the Internet Search Path. Component Download will *not* continue searching for newer versions of components.

Uses for Internet Search Path

The Internet Search Path can be used in two ways:

1. Object Store servers at the beginning of the path will be asked for code *before* checking the location specified in the szCodeURL parameter for CoGetClassObjectFromURL. Servers at the beginning of the search path will thus be checked *before* trying the location specified in the CODEBASE attribute of an <OBJECT> tag. This is a useful feature for corporate intranets, because it allows intranet administrators to set up a local Object Store that is used to serve code for download by employees. (in fact, it is possible to disable the CODEBASE attribute for <OBJECT> tags by removing the CODEBASE keyword from the search path.
2. "Object Store" servers at the end of the search path will be asked for code *after* trying the location specified in the szCodeURL parameter for CoGetClassObjectFromURL, and thus *after* trying the location specified in the CODEBASE attribute. This allows registration of default Object Store locations on the World Wide Web, where browsers can find code when no CODEBASE location is explicitly specified.

Internet Search Path without HTTP

The Internet Search Path assumes that all Object Stores on the search path are "Active" HTTP servers capable of handling HTTP POST requests and querying an object database. In future revisions, it is planned to allow Object Stores on FILE or FTP servers (or simple HTTP servers) in addition to the existing support for "Active" HTTP servers. No further details are available."Pluggable" Setup-script handlers

Although Internet Component Download currently supports a limited .INF setup-script syntax, future releases will take into consideration the need to support "hooks" that allow custom setup handlers to interact with the component download and installation process. For example, it would be desirable to use Win32 standard SetupX .INF files for installation. However, such scripts are not "safe", and allowing such installations would require signing of "trusted": setup scripts. Such work is being considered for future versions of Internet Component Download. No further details are available at this time.

Needs that aren't met by Internet Component Download

There are various situations in which code needs to be downloaded with trust verification but the code is not an OLE Object. Such cases are not addressed by the current specification of the Internet Component Download mechanism. Solutions for these cases need to use the WinVerifyTrust mechanism directly, as detailed below:

- **<A HREF> tag in HTML:** It is possible in HTML to download and run .EXE files directly using the <A HREF> tag. The HTML parser uses URL Moniker directly to download this code, and it calls WinVerifyTrust to check validity.
- **Scripts:** scripting languages will need to define a mechanism for inserting certificates in the script (perhaps in special comments). Given such a mechanism, the WinVerifyTrust service will provide trust verification of any such scripts that are downloaded from the internet.
- **Full applications, other:** The existing Internet Component Download will not handle extremely complex download situations (e.g. download/install DOOM, register device drivers, reboot machine). Future releases will aim to allow hooking into the Component Download mechanism to provide more complicated setup routines.

Appendix - Registry Details

The Internet Component Download service will keep registry entries for every new downloaded component. These registry entries will be useful for (a) writing a utility for cleaning up the code storage, or (b) migrating the Component Download service to use a code cache rather than a permanent store(14).

Why the existing "SharedDLL" mechanism is inadequate

To do correct code caching, the existing shared DLL ref. counting scheme will not suffice, because ref. counts are easily inflated. Specifically, any application that is re-installed increases the ref. count on a shared DLL even though that DLL already has a ref. count belonging to the particular application. (this is already broken for current ref. counting, but it will especially fail for Code Download, in which OCXs are used by multiple pages quite regularly, and there is no way of knowing which OCXs need reference counts.

The new "ModuleUsage" mechanism in the registry for tracking usage of shared components.

To do ref. counting correctly, Component Download will maintain a ModuleUsage section in the registry which holds a list of "owners" and "clients" for each shared module. Thus the registry can keep track of *who* is using a shared module, not just *how many clients* that module has. The registry entries would use the following syntax:

```
[ModuleUsage]
  [<Fully Qualified Path&File Name>]
    .FileVersion=a,b,c,d
    .Owner = Friendly Name/ID of Owner
    <Client ID > = <info peculiar to this client>
    <Client ID > = <info peculiar to this client>
```

A ModuleUsage section in a sample registry would look something like the following:

```
Under My
Computer\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion:

[ModuleUsage]
  [c:\windows/system/mfc40.dll]
    .FileVersion=0,4,0,0
    .Owner = Microsoft Internet Code Downloader
    Microsoft Internet Code Downloader= <any info, or default>
    AnotherAppID= <any info, or default>
```

Key name	Description
<Fully Qualified Path&File Name>	This is the full path of the shared module. This name has to use "/"s instead of "\"s because the "\" is an invalid char in a key name.
.Owner	The application that installs the shared module and creates the original ModuleUsage section will put some identifier in the Owner key section. If the DLL already existed on the system then and this Module Usage key did not exist then the .Owner key should be set to "Unknown" and the DLL should not be removed on uninstall. The owner should always also enlist itself as a client.
.File Version	The version number for the shared module.
<Client ID>	ID of a client who is using the shared module. The value corresponding to each client key contains client specific information. When the client is Internet Component Download, the <Client ID> is "Microsoft Internet Code Downloader", and the client-specific information is a number which serves as a reference count. For other clients, the client-specific information should be the full path of the client, so that if the client is accidentally deleted it is possible to do garbage collection.

Every client of this module is expected to increment and decrement the existing SharedDLLs section in the registry as well (a client only increments this value once when it adds itself as a client under [ModuleUsage]). This is to allow a migration path for apps currently implementing only SharedDLLs scheme.

This registry information complements the reference counts in the SharedDLLs section by

remembering which clients are actually using a shared module. This counting scheme will work correctly and allow caching of downloaded code. Furthermore, when downloading files, Internet Component Download can use this registry information as an efficient shortcut for verifying whether a file needs to be overwritten because it is an out-of-date version.

Footnotes

(1) Note: Internet Component Download as specified will not download anything other than OLE Objects. This document does not list steps needed to download/certify other entities. For other code-download needs see documentation for WinVerifyTrust.

(2) In future releases code for Document Object components will likewise be downloaded and installed automatically.

(3) Note: The <OBJECT> tag used to be called the <INSERT> tag. This change was decided on by the W3C on 2/13.

(4) Care must be taken so that the cabinet file contains only those files that must *necessarily* be downloaded (e.g. the OCX executable itself). Any additional helper DLLs (e.g. MFC) may have already been installed and if so should not be bundled into the cabinet.

(5) Note that Internet Component Download makes the assumption that a *newer* version of an object with the same ClassID is *always backwards compatible* with previous versions. A newer version of an object may be used to replace older versions without worry of losing functionality. If a newer version of an object is not backwards compatible with previous versions, it is advised to assign a new ClassID to the incompatible implementations in order to avoid one overwriting the other resulting in loss of functionality.

(6) If Internet Search Path is used to find the *latest* version of an object, Component Download will search the path, querying servers, and it will use the *first matching component* that is a newer version than the existing version installed on the system (if any). For more details see the below documentation on Internet Search Path.

(7) Note: The MIME scheme described here is *temporary*. Obviously this scheme results in too many MIME types. Eventually, MIME attributes will be used for the purpose of describing platform-dependent code (e.g. application/x-cabinet; os=win32 cpu=x86). Until more HTTP servers support such requests, the temporary scheme described above should suffice.

(8) Internet Component Download accomplishes self-registration using the /regserver command-line argument for .EXE files, and DLLRegisterServer() for other executables (.DLL, .OCX)

(9) See the specification for further details.

(10) Actually, this UI is displayed by the WinVerifyTrust mechanism that is used within Component Download.

(11) This directory location is hard-coded for initial releases. In future releases users may use a registry setting or a Control Panel applet to choose this directory. Component code will be installed in this directory unless a previous version exist. In such cases, the Component Download mechanism will attempt to replace the previous version and invoke ICodeInstall::OnCodeInstallProblem.

(12) Currently Object Stores must be HTTP servers that can serve content dynamically, for instance via ISAPI. In future versions a mechanism will be introduced allowing non-HTTP Object Stores.

(13) Note: an HTTP **POST** request is used, not a GET request. This is because the number of parameters involved is large enough that a GET request may exceed the maximum URL length of 1024 characters.

(14) Either of these would be intelligent about un-installing and un-registering component code using its existing self-registration mechanism.

URL Open Stream Functions

"...pass in a URL, get a stream..."

Without a doubt the easiest, most efficient and most powerful way to download data from the Internet is using the new URL Open Stream (UOS) functions. Before jumping into the specifications, let's cover a few of the high-level features and the philosophy behind the creation of these functions. The UOS functions are the newest addition to the ActiveX™ extensions to the Win32 API. They combine the familiarity of C-style programming with the power of COM. These functions work equally well inside an ActiveX framework (for example, a component, a document or frame window, a subcomponent, or a scriptable object) or completely standalone.

Unlike other more complex interface negotiations, using these functions requires knowledge of no more than two COM interfaces, **IStream** and IBindStatusCallback.

Because these functions use services from URL monikers and WinInet, all the caching and thread synchronization features of those components are in use whenever you call these functions. In addition, the UOS functions handle all the host binding operations if your code is in an ActiveX container: the UOS functions automatically do the right thing to manage the download in the ActiveX client framework.

Every UOS function works in the same basic way: the caller implements an IBindStatusCallback interface (optional in some cases), then calls the function. The URLOpenStream and URLOpenPullStream functions require the caller to be on a thread that has a message loop **(GetMessage/DispatchMessage)**. In the case of an ActiveX component, a message loop is a given if this function is called from the main thread. For a standalone application without a user interface, a message loop is still necessary for these functions.

With the URL Open Stream functions, you can:

- Download a URL to a file with a single function call. You can optionally get progress notifications in the background.
- Create a blocking-type stream with a single function that will block when you call **IStream::Read**. You can optionally get progress notifications in the background.
- Post an Internet query (using the HTTP POST verb) with a single call and get the results in a stream.
- Optionally hook into the ActiveX client framework simply by passing your *this* pointer.
- Configure callbacks using either the push or pull model.

IBindStatusCallback

Callbacks are issued on an **IBindStatusCallback** interface that is implemented on the caller. This is a simple callback interface to implement, because most of the methods on the interface are optional (for example, **OnStartBinding**, **GetPriority**, **OnStopBinding**) and can simply return S_OK or E_NOTIMPL. Although clients may choose to implement many of the **IBindStatusCallback** methods, the only callback required to do anything for the UOS functions to work correctly is **OnDataAvailable** for the URLOpenStream and URLOpenPullStream functions. In fact, for some UOS functions (URLDownloadToFile and URLOpenBlockingStream), **OnDataAvailable** is never called because it is unnecessary. Furthermore, GetBindInfois *never* invoked for UOS clients, because the bind information is determined according to which UOS function is being called. The UOS programming model is straightforward because there are no special flags to pass to the functions.

The URL Open Stream functions are described below:

URLOpenStream

URLOpenStream creates a push-type stream object from a URL. The data is downloaded from the Internet as fast as possible. Every time data is available, it is "pushed" at the client through a notification callback.

URLOpenStream(
LPUNKNOWN *pCaller*,
LPCWSTR *szURL*,
DWORD *dwResv*,
LPBINDSTATUSCALLBACK *lpfnCB*
);

Parameters

LPUNKNOWN *pCaller*
Pointer to the controlling **IUnknown** of the calling ActiveX component (if the caller is an ActiveX component). If the caller is not an ActiveX component, this value may be set to NULL. Otherwise, the caller is a COM object that is contained in another component (such as an ActiveX Control in the context of an HTML page). The parameter represents the outermost **IUnknown** of the calling component. The function will attempt the download in the context of the ActiveX client framework and allow the caller's container to receive callbacks on the progress of the download.

LPCWSTR *szURL*
The URL to be converted to a stream object. Cannot be NULL.

DWORD *dwResv*
Reserved for future use; must be zero.

LPBINDSTATUSCALLBACK *lpfnCB*
Pointer to caller's IBindStatusCallback interface, on which **URLOpenStream** calls **OnDataAvailable** every time data arrives from the Internet. **OnDataAvailable** can return E_ABORT to abort the download. When the callback is invoked and the *pstm* member of the STGMEDIUM structure is not NULL, the caller can read from the stream the amount of data specified in the *dwSize* argument passed with the **OnDataAvailable** call (delta anything that has been read on previous calls to **OnDataAvailable**). If the caller does not read the full amount or does not call *pstm*-> **Read** at all, **OnDataAvailable** will still be called the next time data arrives, as long the *grfBSCF* flags do not indicate BINDF_LASTDATANOTIFICATION. In that case, no more data will be downloaded. Any data that is not read at any given time will still be available the next time **OnDataAvailable** is called.

Remarks

This function will always try to obtain the bits from the local cache, if available. If the data was not originally in the cache, URLOpenStreamwill also always try to put the bits that are downloaded into the cache.

The logic in the following code fragment is a typical implementation of OnDataAvailableas it is used by the URLOpenStreamfunction:

```
HRESULT MyBindStatusCallback::OnDataAvailable
(
  DWORD grfBSCF, DWORD dwSize, ..., STGMEDIUM * pstgmed
```

```
)
{

  if( dwSize &lt sizeof(BITMAPINFOHEADER) )
    return(NOERROR); // not enough has been read yet, just return

  if( !g_bGotInfoHeader ) // did we get info before?
  {
    // No, go ahead, read now...

    DWORD dwRead;
    HRESULT hr = pstgmed->pstm->Read( &bmih, sizeof(bmih), &dwRead);
    if( SUCCEEDED(hr) )
    {
    // now we got it... we can return
    g_bGotInfoHeader = TRUE;
    return(hr);
    }

  }
}
```

URLOpenBlockingStream

URLOpenBlockingStream creates a blocking-type stream object from a URL. The data is downloaded from the Internet on demand by a call to **IStream::Read**. The **Read** call will block until enough data has arrived.

URLOpenBlockingStream(
LPUNKNOWN *pCaller*,
LPCWSTR *szURL*,
LPSTREAM **ppStream*,
DWORD *dwResv*,
LPBINDSTATUSCALLBACK *lpfnCB*
);

Parameters

LPUNKNOWN *pCaller*
Pointer to the controlling **IUnknown** of the calling ActiveX component (if the caller is an ActiveX component). If the caller is not an ActiveX component, this value may be set to NULL. Otherwise, the caller is a COM object that is contained in another component (such as an ActiveX Control in the context of an HTML page). The parameter represents the outermost **IUnknown** of the calling component. The function will attempt the download in the context of the ActiveX client framework and allow the caller's container to receive callbacks on the progress of the download.

LPCWSTR *szURL*
The URL to be converted to a stream object. Cannot be NULL.

LPSTREAM * *ppStream*
Pointer to the **IStream** interface pointer on the stream object created by this function. The caller can read from the stream as soon as it has this pointer. If the data requested has not yet been downloaded, the **Read** method will block until enough data has been downloaded. The following is a code fragment that logically does this:

```
IStream * pStream;
URLOpenStream( 0, L"http://www.msn.com/", &pStream, 0, 0);

char buffer[0x100];

DWORD dwGot;
HRESULT hr = NOERROR;

do {
  hr = pStream->Read( buffer, sizeof(buffer), &dwGot );

  .. do something with contents of buffer ...

} while( SUCCEEDED(hr) );
```

DWORD *dwResv*
Reserved for future use; must be zero.

LPBINDSTATUSCALLBACK *lpfnCB*
Pointer to the IBindStatusCallback interface pointer on the caller. **URLOpenBlockingStream**

calls this interface's **OnProgress** method on some connection activity, including the arrival of data. **OnDataAvailable** is never called. Implementing **OnProgress** allows a caller to implement some user interface or other progress monitoring functionality. It also allows the download operation to be completely canceled by returning E_ABORT from the OnProgresscall. Can be NULL.

Remarks

This function will always try to obtain the bits from the local cache, if available. If the data was not originally in the cache, **URLOpenBlockingStream** will also always try to put the bits that are downloaded into the cache.

URLDownloadToFile

URLDownloadToFile downloads bits from the Internet and saves them to a file. The client can optionally be notified of progress via a notification callback.

URLDownloadToFile(
LPUNKNOWN *pCaller*,
LPCWSTR *szURL*,
LPCTSTR *szFileName*,
DWORD *dwResv*,
LPBINDSTATUSCALLBACK *lpfnCB*
);

Parameters

LPUNKNOWN *pCaller*
Pointer to the controlling **IUnknown** of the calling ActiveX component (if the caller is an ActiveX component). If the caller is not an ActiveX component, this value may be set to NULL. Otherwise, the caller is a COM object that is contained in another component (such as an ActiveX Control in the context of an HTML page). The parameter represents the outermost **IUnknown** of the calling component. The function will attempt the download in the context of the ActiveX client framework and allow the caller's container to receive callbacks on the progress of the download.

LPCWSTR *szURL*
The URL to be downloaded. Cannot be NULL.

LPCTSTR *szFileName*
Name of the file to create with bits that come from the download.

DWORD *dwResv*
Reserved for future use; must be zero.

LPBINDSTATUSCALLBACK *lpfnCB*
Pointer to the IBindStatusCallback interface pointer on the caller. URLOpenBlockingStream calls this interface's **OnProgress** method on some connection activity, including the arrival of data. **OnDataAvailable** is never called. Implementing **OnProgress** allows a caller to implement some user interface or other progress monitoring functionality. It also allows the download operation to be completely canceled by returning E_ABORT from the **OnProgress** call. Can be NULL.

Remarks

This function will never try to obtain the bits from the local cache, even if available. In addition, it will never try to put the bits that are downloaded into the cache.

URLOpenPullStream

URLOpenPullStream creates a pull-type stream object from a URL. The data is downloaded from the Internet on demand. If not enough data is available locally to satisfy the requests, the **IStream::Read** call will not block until enough data has arrived. Instead, **Read** will immediately return E_PENDING, and **URLOpenPullStream** will request the next packet of data from the Internet server.

URLOpenPullStream(
LPUNKNOWN *pCaller*,
LPCWSTR *szURL*,
DWORD *dwResv*,
LPBINDSTATUSCALLBACK *lpfnCB*
);

Parameters

LPUNKNOWN *pCaller*
Pointer to the controlling **IUnknown** of the calling ActiveX component (if the caller is an ActiveX component). If the caller is not an ActiveX component, this value may be set to NULL. Otherwise, the caller is a COM object that is contained in another component (such as an ActiveX Control in the context of an HTML page). The parameter represents the outermost **IUnknown** of the calling component. The function will attempt the download in the context of the ActiveX client framework and allow the caller's container to receive callbacks on the progress of the download.

LPCWSTR *szURL*
The URL to be converted to a stream object. Cannot be NULL.

DWORD *dwResv*
Reserved for future use; must be zero.

LPBINDSTATUSCALLBACK *lpfnCB*
Pointer to caller's IBindStatusCallback interface, on which **URLOpenPullStream** calls **OnDataAvailable** every time data arrives from the Internet. **OnDataAvailable** can return E_ABORT to abort the download. When the callback is invoked and the *pstm* member of the STGMEDIUM structure is not NULL, the caller can read from the stream the amount of data specified in the *dwSize* argument passed with the **OnDataAvailable** call (delta anything that has been read on previous calls to **OnDataAvailable**). If the caller does not read the full amount or does not call *pstm->* **Read** at all, **OnDataAvailable** will not be called again until this happens and Readreturns E_PENDING.

Remarks

This function will always try to obtain the bits from the local cache, if available. If the data was not originally in the cache, **URLOpenPullStream** will also always try to put the bits that are downloaded into the cache.

The pull model is slightly more cumbersome than the push model, but it gives total control to the client over the amount of Internet access for the download.

The logic in the following code fragment is a typical implementation of **OnDataAvailable** as it is used by the **URLOpenPullStream** function:

```
HRESULT CMyBindStatusCallback::OnDataAvailable( ...)
{
```

```
  HRESULT hr = NOERROR;
  DWORD dwAmountToRead = dwSize - g_readSoFar;
  BYTE * buffer = new BYTE[ dwAmountToRead ];

  while( TRUE )
  {
    DWORD dwRead;

    hr = pstgmed->pstrm->Read( buffer, dwAmountToRead, &dwRead );

    if( hr == E_PENDING )
    {
      // we'll get notified again when more data comes
      return(NOERROR);
    }

    if( SUCCEEDED(hr) )
    {
      // ok, process bits ....
      // keep looping
    }
    else
    {
      // we have an error...
      return(hr);
    }
  }
}
```

URLOpenHttpStream

URLOpenHttpStream is a catch-all function for doing more sophisticated HTTP and FTP downloads, such as performing an HTTP POST. By filling in the UOSHTTPINFO structure, you inform the function of the functionality you expect and how you want the bits to be delivered.

The UOSHTTPINFO structure is defined as follows:

URLOpenHttpStream(
LPUOSHTTPINFO * *lphttpInfo*
);

Parameters

ULONG *ulSize;*
// Size of this structure
LPUNKONWN *punkCaller;*
// Same as pCaller in previous functions. Can be NULL. LPCTSTR *szURL*;
// URL to be downloaded
LPCTSTR *szVerb*;
// One of "GET", "PUT", "POST", or a custom
// verb understood by the server. If NULL,
// "GET" is assumed.
LPCTSTR *szHeaders*;
// HTTP headers to use during connection to server.
// Can be NULL.
LPBYTE *szPostData*;
// Additional data to send after the headers.
// Typically this will be "POST" data parameters.
// Can be NULL.
ULONG *ulPostDataLen*;
// Size of *szPostData*. Must not be 0 if
// *szPostData* is not NULL.
ULONG *fURLEncode*;
// Flags with either the UOS_URLENCODPOSTDATA,
// UOS_URLENCODEURL, or can be 0. The two
// flags can be combined by a bitwise OR.
ULONG *ulMode*;
// Can be one of UOSM_PUSH, UOSM_PULL, UOS_BLOCK, or
// UOS_FILE. Each of these maps to one of the
// functions above and makes this function's
// programming model fit the corresponding function.
LPCTSTR *szFileName*;
// Name of the local file to write URL data to if *ulMode*
// is UOS_FILE. Otherwise, must be NULL.
LPSTREAM **ppStream*;
// Pointer to IStream pointer if *ulMode* is UOS_BLOCK.

// Otherwise, must be NULL.
LPBINDSTATUSCALLBACK *lpbscb*;
// Pointer to IBindStatusCallbackinterface. This interface
// and the caller's programming model responsibility
// depend on the *ulMode* flag above. The function
// will behave exactly like the corresponding
// UOS functions above, depending on the flag settings.

Remarks

Depending on the values in the UOSHTTPINFO structure, this function may try to obtain the bits from the local cache, if available. Similarly, if the data was not originally in the cache, the function may try to put the bits that are downloaded into the cache. There are cases where this does not happen: for example, it will not do either of these in the value of *szVerb* is "POST".

Disclaimer

The Licensing Problem: Background, Goals, Assumptions

In order to create a market for ActiveX Controls to proliferate on the World Wide Web, it's essential to standardize a method for licensing ActiveX Controls embedded in HTML documents. Before discussing proposed licensing schemes, we establish criteria and objectives for such schemes.

One must first understand the existing **IClassFactory2** (ICF2) mechanism for licensing OLE Controls on the desktop-i.e. for use in Visual Basic. In the existing scheme, a VB programmer buys a *design-time license* for a control. When the programmer creates a VB form, the authoring tool embeds a free *run-time license* for the control inside the form, allowing anybody to use the form without paying for the control. The control itself participates in this process via the **IClassFactory2** interface, which exposes two important methods. The first method, RequestLicKey, is used by the authoring tool (VB) for querying the *run-time license* at *design-time*. The second method, CreateInstanceLic, is used by the run-time application (VBRUN.DLL) for instantiating the control using this *run-time* license. For more information on the existing scheme for licensing of Controls, see OLE Reference documentation for the **IClassFactory2** interface.

Having understood the desktop model for licensing controls, we specify objectives for online licensing of controls. The goals for online licensing make this model quite analogous to the desktop model:

- **Business model:** web authors purchase a design-time license to author HTML pages using a control. The run-time license for browsing the HTML page is typically free.
- **Security:** although a 100% foolproof mechanism would be great, it's not necessary. "Standard" online licensing need only be secure enough to prevent casual or unintentional copying of licensed controls. It's therefore not necessary to make piracy *impossible*-it's adequate to make piracy *inconvenient*. A control developer can always develop a private scheme that is more secure.
- **Convenience for authors:** licensing should not make things difficult for web authors. It's clear that the harder it is for web authors to use licensed controls, the less chance they will actually pay for such controls.
- **Compatibility:** the online model should accommodate existing controls that use the **IClassFactory2** licensing mechanism. This means each HTML page should have some associated control licensing information that is created at design-time and recognized by web browsers at run-time. In this model an HTML page is analogous to a Visual Basic form.
- **Versatility:** the online model must be versatile enough to support other online ActiveX Control containers, such as the NCompass ActiveX plug-in.

The Internet Explorer 3.0 Licensing Scheme

Online licensing of controls will continue to use the **IClassFactory2** mechanism that is implemented by existing licensed controls. An HTML page with licensed controls requires a single associated *license package* which stores the run-time licenses for all the controls used on the page. The license package (.LPK file) stores an array of (CLSID, license) tuples. The HTML page points to the license package via a *relative* URL reference inside the HTML. At design-time, authoring tools or utilities should use **IClassFactory2::RequestLicKey** to create the .LPK file and the associated HTML element (tag) that refers to it. At run-time, web browsers should extract the necessary run-time licenses from the license package in order to instantiate licensed controls embedded in the HTML page using **IClassFactory2::CreateInstanceLic**. In this model the HTML page is analogous to a Visual Basic form, HTML-authoring tools or helper utilities play the role of the VB Design Environment, and the web browser plays the role of the VB run-time (VBRUN.DLL).

It is important to note that the URL reference to the license package (.LPK) must be *relative only*. This makes it inconvenient to pirate licensed controls. Certainly it's less possible to accidentally pirate controls, because one can't copy an HTML page with a relative URL to a .LPK file and expect the page to still work. Since the URL to the .LPK file must be relative, in order to pirate a control one would have to copy the .LPK file knowingly. By using a plain-text format for the .LPK file, it is possible to include a legal copyright statement at the top of the file, dissuading anybody who downloads this file in order to copy it to a pirated site.

Web browsers can support this licensing scheme by posing additional obstacles for pirates who wish to copy .LPK files - for example making it difficult to download and save such files, or obscuring the cached copies of such files. Such obstacles do not make piracy impossible, but they make it even more inconvenient. Implementation of such obstacles is **optional** for web browsers, and is not a requirement of the proposed licensing scheme.

Justification of the Proposed Scheme

To justify the proposed scheme, we present a brief outline of the advantages, as well as a brief summary of alternatives that were ruled out for various reasons.

Advantages of the Scheme:

- **Convenience for web authors** - while a sophisticated authoring tool could make the design-time process completely invisible for web authors, even a simple design-time utility could make creation of .LPK files quite easy for authors that write HTML directly using tools such as Notepad.
- **Extra convenience for web sites** - a site with many web pages and many licensed controls could easily create one .LPK license package file containing all the licenses for all the pages on the site. After the one-time creation of this .LPK file, it is easy to refer to it from all pages on the site.
- **Inconvenience for pirates** - the average web developer cannot accidentally pirate licensed controls without realizing that they are breaking copyright laws. Although intentional piracy is not impossible, it is rather inconvenient and requires knowledge of the .LPK file format.

Other Licensing Alternatives that Were Considered

- **Tying the .LPK file to a particular site(s)** - one could achieve additional security by tying a license package to the site at which it would be used (for example, by including a list of URL prefixes or an X.509 certificate in the .LPK). However, the additional security is not worth the inconvenience to web authors, especially for publishers that host many sites, or for HTML-designer contractors that write HTML pages for many different publishers.
- **Tying the .LPK to a particular HTML page** - it is possible to use CRC digests to link a .LPK to a particular .HTML page. This was widely rejected as being too difficult at design time, and impossible for dynamically generated content.
- **Using digital certificates for foolproof security** - although in the future Microsoft intends to pursue 100% foolproof solutions to the licensing probleme, for short-term needs (Internet Explorer 3.0) a simpler solution is much more practical.

Technical Details

HTML Syntax for Referencing .LPK Files

License packages must be inserted in HTML files using the existing <OBJECT> tag (element). The license package is interpreted by a *License Manager* object that is used to hand licenses to other Controls on the HTML page. The license package object must be embedded in the HTML page *before* any other objects that require licensing. The specific syntax for embedding license packages in HTML is as follows:

```
<OBJECT CLASSID = "clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
     <PARAM NAME="LPKPath" VALUE="relative URL to .LPK file">
</OBJECT>
```

Note Internet Explorer 3.0 will only honor the *first* license package in an HTML page. If controls on this page require licenses that are not inlcuded in this license package, then they will afil to instantiate.

License Package File Format

The .LPK license package file is a plain text file, and must be labeled by servers with the MIME type *text/plain*. Any binary data in the file is *uuencoded*. This allows the file to be viewable by web browsers, so that anyone trying to copy the file would clearly notice the copyright statement at the top of the file. The contents of the file are as defined below. This file will be interpreted by a *License Manager* object.

.LPK Header	**This header identifies the file type: "*LPK License Package*"**
Copyright text or other legal statement	"Legalese" to dissuade casual copying of .LPKs.
LPK version GUID	In plain-text on a line by itself. This GUID is used to mark the beginning of the real license package data, and it is also used to identify the LPK file format version: "*{5220cb21-c88d-11cf-b347-00aa00a28331}*"
Uuencoded(Base64) license package:	

```
struct {
    UUID uuidLPKFile;   //
unique per LPK
    DWORD dwLicenses;   //
number of licenses in the
file
    LICENSEPAIR
aLicenses[]; // array of
license pairs
} LICENSEPACKAGE;

struct {
    CLSID clsid;        //
clsid of object
    DWORD cchLic;       //
Number of characters in
the license
    WCHAR ach[];        //
License (saved as UNICODE
characters)
} LICENSEPAIR;
```

External Dependencies

Authoring Tools

HTML authoring tools will make it easy to create HTML pages with embedded ActiveX Controls. Such tools should be responsible for creating the .LPK license package for licensed controls used on a page or on a web site. Since there may be a one-to-many mapping between LPKs and HTMs, this may be more difficult for pagebased authoring tools as opposed to web-based authoring tools (e.g. Microsoft FrontPage).

Clearly, tool support is necessary for Notepad HTML authors as well. The solution is a simple GUI tool that lists all controls that are installed on a machine with *design-time* licenses. The tool allows a user to create a .LPK license package by selecting which Controls should be included in the package. A second tool could parse HTML pages an create a .LPK file for all the controls that require licensing.

NCompass support for licensing ActiveX Controls

It is crucial that the licensing scheme described above works in other ActiveX-enabled web browsers, particularly in the NCompass ActiveX Plug-in. Because the plug-in specification only allows plug-ins to be specified using the <EMBED> syntax, therefore pages authored to work with both Internet Explorer 3.0 *and* NCompass will need to use syntax similar to the following:

```
<OBJECT CLASSID = "clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="relative URL to .LPK file">

    <EMBED SRC = "FOO.LPK">
</OBJECT>
```