OLE Hyperlinks

December 1995

Distribution: Public

DRAFT

Hyperlinking and in-frame navigation are central parts of the Internet user-interface model. This document contains the specification for OLE Hyperlinks, a set of system services and integration interfaces that, along with OLE Document Objects, allow applications to integrate their new and existing applications with each other and with Web browsers.

1 OLE Hyperlinks

Introduction

One of the most compelling ease-of-use features in World Wide Web applications is the navigation-based user-interface model including ***point-and-click hyperlink navigation***, a ***history*** list (with commands like *Go Back* and *Go Forward*), and a ***favorites*** list. Like other OLE-based application integration technologies, OLE Hyperlinks enable you to offer this same user-interface and navigation model within your documents and application, allowing you to integrate them seamlessly with other hyperlinking applications including web browsers.

Specifically, OLE Hyperlinks allow you to:

- Add hyperlinking support to existing documents, objects, and applications.
- Integrate the documents of your application into enabled World Wide Web browsers.
- Support and integrate with key features of OLE Hyperlink applications (*History, Favorites, Go Back, Go Forward*, *Go Home*)
- Support navigation to documents on the Internet or any large-scale public network characterized by high latency.

This document provides an overview of the OLE Hyperlinking architecture and then describes the interfaces needed to add hyperlink support to an application. Not all the details described are necessary for participating in OLE Hyperlinking−applications can choose to support various degrees of integration with the OLE Hyperlinking architecture. Where applicable, this document describes how to provide minimal or "simple" hyperlinking support, and also how to extend this support for more complete hyperlinking integration.

Related Document	**Location**
HREF="hlsimple.doc	hlsimple.doc

Note: Many of the details of OLE Hyperlinking are still in flux. Application developers are currently encouraged to use the "Simple Hyperlink Navigation" APIs (see above) to add hyperlinking support to their documents and applications. The rest of the architecture and the interfaces specified below are still subject to change as the implementation is being finalized. Clients of "Simple Hyperlink Navigation" will continue to work using the full OLE Hyperlinking architecture once it is final.

What is a Hyperlink?

A *hyperlink* is a reference to another location - within the existing application/document or in a new application/document[1]. The data of the document/object can be stored in file-system files, machines on the Internet (referred to by URLs), or any arbitrary location that can be referenced via OLE monikers. The hyperlink reference is a *(target, location)* tuple, stored as a moniker for the *target* part and a string for the *location* part. The navigation to the location can be done by binding to the *target* and then asking it to navigate to the *location*.[2]

[1] Specifically, a hyperlink may be a reference to a location in (1) the same document/object that contains the hyperlink, (2) a different (top-level or embedded) object/document of the same class, or (3) a different object/document of a different class.

[2] There is a deliberate avoidance of using a single moniker for the hyperlink reference. The drawback with the single moniker approach is that the binding results in object creation, one per each location we navigate to. The object creation introduces all the problems associated with the life time management in addition to incurring the performance overhead (overhead of marshaling one extra object in cross-process scenarios, etc.) and memory overhead. With this referencing mechanism, there is no object creation for the location if the hyperlink is used for navigation only. Hyperlinks are different than regular OLE or DDE links. In the case of OLE or DDE links, the link resolver would want to maintain a connection to the link source, and the maintenance of this connection introduces issues of object life-time management. On the other hand, the system hyperlink "resolver" is interested mainly in navigation to the hyperlink *target*. By avoiding the binding to the destination object, we simplify the hyperlinking interfaces and implementations. In a very high percentage of cases hyperlinks are used for navigation only, and hence there won't be any need for hyperlink resolver to keep an (interface) reference to the hyperlink. The object stays running either because it is visible to the user and/or because the hyperlink *browse context* keeps a reference to it.

How a hyperlink is presented to the user is up to the hyperlink *container* and the context of the hyperlink. It is common for hyperlinks to be presented as colored, underlined text, as hotspot regions on an image, or as pushbuttons. However, there are no user interface requirements limiting the presentation of hyperlinks, although guidelines suggest that they should be made obvious – via coloring, underlining, or by changing the cursor or displaying tooltips when moused over.

What is Hyperlink Navigation?

Hyperlink navigation involves a transition from one document/object/application, known as the hyperlink *container*, to another document/object/application, known as the hyperlink *target*[3]. Usually both remain running, but the hyperlink target visually replaces the hyperlink container. Hyperlink containers and targets may be top-level documents or OLE Document Objects (DocObjs) in a browser application. Because of this variation, there are many possible forms of navigation.[4]

1. *From one top-level document to another top-level document*. (in the absence of a browser).
2. *From a top-level document to an OLE DocObj in a browser* (e.g. navigation from a stand-alone application to an HTML document).
3. *From one OLE DocObj in a browser to another OLE DocObj in the same browser.* (e.g. navigation from one HTML document to another)
4. *From an OLE DocObj in a browser to an OLE DocObj in a Binder-like application*. (e.g. if the hyperlink target is embedded in an Office Binder document).[5]
5. *From one location in a object/document to another location in the same object/document.* This is applicable to all flavors of hyperlink containers / targets.

As you can see, the window gets reused only in cases 3 and 5. In the other cases, the hyperlink target appears in a new window. The OLE Hyperlinks architecture suggests a mechanism for creating the *illusion* of window reuse in such cases: the hyperlink container passes its current window position to the hyperlink target, and the hyperlink target positions its window in the exact same location. Upon successful navigation the hyperlink container hides its window.

Browse Contexts

By looking at the possible forms of navigation one can see that multiple objects/documents (perhaps from different processes) share some global context. This context knows the order in which documents have been visited. All jumps are recorded with this context, and this context chains them together in a *navigation stack*, thus knowing where to go as result of *Go Back* or *Go Forward*. This context is called the *Browse Context*.

Even though a browse context is global and spans multiple processes, it need not be global per *User* like *History* and *Favorites*. There can be multiple browse contexts active at once. For example, in a web browser one can start a new browse context using the right mouse pop-up menu item *Open In New Window.*

OLE Hyperlinking defines the interface for the standard system browse context. This allows hyperlink-aware documents and applications to integrate the browse context's navigation stack via standard interfaces defined below.

Architecture Overview

In the following sections you will find the description of various components in the OLE Hyperlinks architecture and the interfaces they implement. Some components below are standard, *system-provided* objects, whereas others are *user-defined* components that participate in OLE Hyperlinking because they implement the appropriate interface(s).

---

[3] Often the hyperlink *target* also serves as a hyperlink *container*. For the sake of clarity, a document or object is henceforth referred to as a hyperlink *container* when it acts as origin of hyperlink navigation, and as a *target* when it is navigated to.

[4] Several describable cases are not listed here as they actually coincide with one or more of the listed cases.

[5] In this scenario the 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.

The "simple" hyperlinking API functions.

OLE Hyperlinking provides many useful functions and services that are needed for general purpose uses and for complete application integration. However, many simple applications are interested only in navigating to a hyperlink target[6]. For simple navigation needs, there are a number of "simple hyperlinking" APIs: HlinkSimpleNavigateToString, HlinkSimpleNavigateToMoniker, HlinkNavigateString, HlinkNavigateMoniker, HlinkGoBack, HlinkGoForward, that allow hyperlink navigation without knowledge of any other hyperlink interfaces or objects. The more complex objects and interfaces described below are useful for more complex needs, such as supporting navigation to a sub-location within a new document type, or allowing cut/paste or drag/drop of hyperlinks.

Note: As described above, many details of OLE Hyperlinking are still in flux. Developers are encouraged to use the "simple hyperlink navigation" APIs above to add hyperlinking support to their documents and applications. The rest of the architecture and the interfaces specified below are still subject to change as the implementation is being finalized.

Hyperlink Target

A *hyperlink target* is a destination of hyperlink navigation. This can be a persisted OLE object that exposes IHlinkTarget, a persisted OLE object that exposes IOleObject, or any file that is viewed when its viewer application is launched via ShellExecute(). An object (document) that wants to be targeted by hyperlinks can choose to implement all or part of the IHlinkTarget interface to integrate tightly with OLE Hyperlinks. If the object does not support IHlinkTarget, it can still act as a hyperlink target, but it won't be able to support internal navigation, and it will not have access to the common *browse context* that holds the navigation stack. A hyperlink target may be a top level container document, or an embedded object of arbitrary nesting, or in general, any object that can be referenced via a moniker.

How to implement an IHlinkTarget

An existing OLE Documents 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. Supporting IHlinkTarget from an OLE Document Object is the recommended way to make sure your document is viewable by browsers and participates in hyperlinking smoothly.

Hyperlink

A *hyperlink* object implements the IHlink interface and encapsulates four pieces of reference information: a moniker to the hyperlink target, a string for the sub-location within the target, a friendly name for the target, and additional parameters. This object completely encapsulates the behavior of navigating to a referenced location. It also supports the ability to save and load itself via IPersistStream, and the ability to be transferred through the clipboard or through drag-and-drop via IDataObject.[7]

A standard hyperlink object implementation is provided with 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 work of navigating, saving, loading, dragging, dropping, cutting, and pasting hyperlinks. Standard hyperlink objects are created via the HlinkCreateFromData, HlinkCreateFromMoniker, HlinkCreateFromString, and OleLoadFromStream APIs. The standard hyperlink object implements the IHlink, IPersistStream, and IDataObject interfaces.

Hyperlink Container and Hyperlink Site

---

[6] For example, a spreadsheet application may wish to allow hyperlinking from individual spread sheet cells. As another example, an OLE Control embedded in an HTML page may wish to navigate to another document.

[7] Note: When saved, the hyperlink object saves the relative moniker in addition to the absolute moniker to the target.

A *hyperlink container* is a document or application that contains hyperlinks[8]. If a hyperlink container (document) uses the hyperlink object (described above) to perform hyperlinking functions, then it may optionally implement hyperlink site objects supporting IHlinkSite for each contained hyperlink.

A *hyperlink site* is used by its corresponding hyperlink to retrieve the moniker of the hyperlink container. This moniker is used to evaluate relative monikers to the hyperlink target. When the relative moniker is NULL, this indicates the target of the link is in the same container object, and IHlink::Navigate can result in an efficient internal jump.[9]

Hyperlink Frame

A *hyperlink frame* is an outer frame that manages one or more hyperlink container documents. An application (e.g. a web browser) that wishes to be a viewer for multiple hyperlink target document types implements IHlinkFrame and manages hyperlink containers in a consistent user interface. Browser applications such as Microsoft® Internet Explorer and the Microsoft® Office Binder are examples of hyperlink frames. Ideally, a hyperlink frame also serves as an OLE Document Object frame, allowing it to support browsing and hyperlinking between various Document Objects.

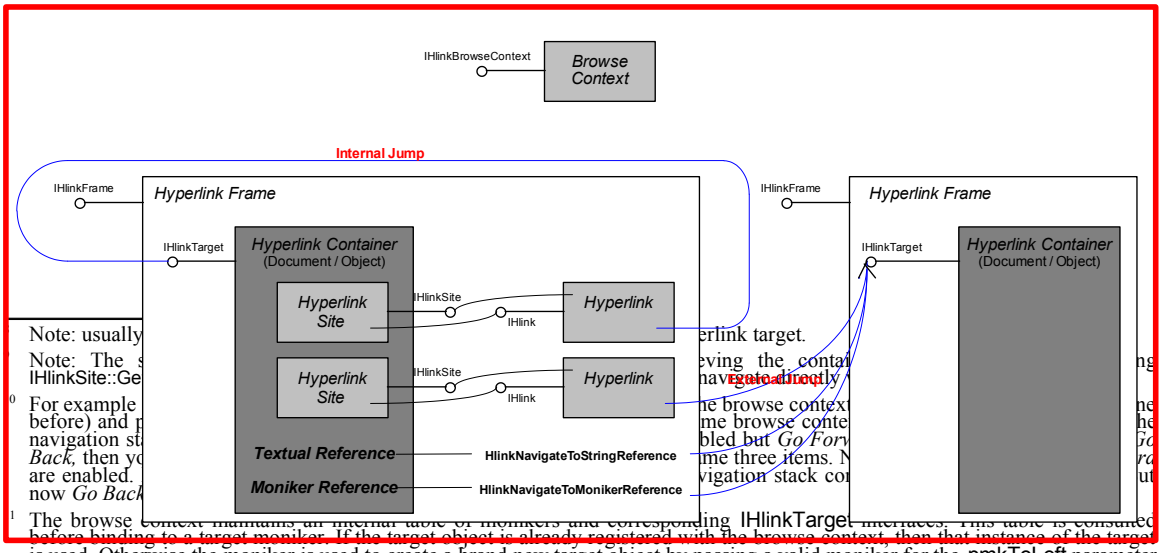
Hyperlink Browse Context

The hyperlink *browse context* maintains the *navigation stack* that is passed around during navigation from one document or application to another. Like the hyperlink object, the browse context is a standard object provided with the system. In addition to maintaining the navigation stack, the browse context knows whether or not *Go Back* and *Go Forward* commands should be enabled.[10] It is also used to pass around window position information so that hyperlinking from one top-level window to another can happen while giving the illusion of window reuse (by positioning the windows on top of each other).

Each hyperlink target is passed a browse context (through IHlinkTarget::SetBrowseContext) the first time it is navigated to. The hyperlink target registers with the browse context (through IHlinkBrowseContext::Register) and holds a reference to it. The target must also notify the browse context each time it is navigated to via IHlinkBrowseContext::OnNavigateHlink. The browse context uses this information to maintain the navigation stack, to remember the "current item" in the stack, and also to manage the lifetimes of registered hyperlink target applications using a MRU (most recently used) scheme[11].

How the pieces connect to each other

The diagram below shows various COM objects involved in OLE Hyperlinking, the interfaces they implement, and the way they are inter-connected:



Note: usually … erlink target.

Note: The … eving the contai… navigate direct…

For example … he browse context … before) and … me browse conte… navigation st… bled but *Go Forv… Back*, then yo… me three items. N… are enabled. … vigation stack co… now *Go Back*

The browse context maintains an internal table of monikers and corresponding IHlinkTarget interfaces. This table is consulted before binding to a target moniker. If the target object is already registered with the browse context, then that instance of the target is used. Otherwise the moniker is used to create a brand new target object by passing a valid moniker for the pmkToLeft parameter to IMoniker::BindToObject. This avoids collision with existing instances of the hyperlink target running via some other means.

Examples

"Simple Hyperlinking"

The navigation in this example takes place between simple hyperlink containers and simple hyperlink targets, none of which understand how to use the hyperlinking browse context. For OLE Controls or OLE Document Objects hosted within a browser such as Microsoft ® Internet Explorer, this is the only hyperlinking support that is necessary to integrate fully in the navigation stack and history list. The following pseudo-code outlines the order or execution of methods among the various objects during navigation.

Note: "simple hyperlinking" is simple mainly for the hyperlink container and the hyperlink target, particularly if both are Document Objects hosted in a DocObj frame. In these cases, the frame takes over all responsibility for providing a navigation stack and integrated history/favorites. Note that there is no definition of "simple hyperlinking" for hyperlink frames.

Starting with a Hyperlink Container

The hyperlink container (e.g. a document or an OLE Control embedded within a document) initiates the hyperlink navigation using one of the "simple hyperlinking" APIs:

```
// do the navigation
HlinkSimpleNavigateToString( "http://www.microsoft.com/foo.htm", NULL, NULL, punkMe, 0, pbc, pbsc, 0);
```

IHlinkFrame::Navigate

The hyperlink frame (e.g. Microsoft Internet Explorer) is called from within the simple navigation API (which packages up the call into a full call to HlinkNavigate). This is the frame's chance to provide a integrated user-interface, progress feedback, cancellation options, and so forth. Often the hyperlink frame simply sets some flags and defers to IHlink::Navigate (sample code provided in "fully integrated" hyperlinking example). Other frames may choose to do more work. For instance, a frame may decide to provide its own IBindStatusCallback in order to listen in on progress notifications during navigation.

IHlink::Navigate

Usually, the system-provided hyperlink object gains control to do the brunt of the navigation work on behalf of the container or frame, resulting in a call to the hyperlink *target*.

IHlinkTarget::Navigate

The hyperlink target then receives control to navigate to the specific location within the target. Notice that support for "sub-locations" is optional. Also, the interpretation of location strings is left to the interpretation of the target object.

```
 STDMETHODIMP
IHlinkTarget::Navigate(DWORD grfHLNF, LPCWSTR szLocation)
{
      IHlinkFrame* phlFrame = NULL;
      // if szLocation is not visible show it

      // if this hyperlink target is an OLE Document Object, try to retrieve the hyperlink frame pointer from the IOleInPlaceFrame
      if (m_poleinplaceframe)
            m_poleinplaceframe->QueryInterface(IID_IHlinkFrame, (void**)&phlFrame);

      // notify the hlink frame and the browse context that the navigation is complete. Note: either phlFrame or m_phlbc may be NULL
      HlinkOnNavigate(phlFrame, NULL, grfHLNF, m_pmk, szLocation, szFriendlyName);

      m_fHide = FALSE;
      m_fHideFrame = FALSE;
}  // IHlinkTarget::Navigate
```

IHlinkFrame::OnNavigate

The hyperlink frame next receives notification of a successful navigation from IHlinkTarget::Navigate in order to reposition its windows and update its windows' visibility. If this is the same frame that hosted the hyperlink container that initiated the navigation, then the flags set below will ensure that the frame remains visible (sample code provided in "fully integrated" hyperlinking example).

"Fully integrated" Hyperlink Navigation

The navigation in the example below takes place between hyperlink containers and hyperlink targets which understand how to use the hyperlinking browse context. The following pseudo-code outlines the order or execution of methods among the various objects during navigation.

Starting with a Hyperlink Container

Before any navigation occurs, the container starts with a hyperlink object that has either been created (using HlinkCreateFromData, HlinkCreateFromMoniker, HlinkCreateFromString) or loaded from persistent data (using OleLoadFromStream). The hyperlink container may initialize the hyperlink through IHlink::SetHlinkSite by passing in an IHlinkSite interface and hyperlink-specific data (dwSiteData), which allows the hyperlink container to use the same hyperlink site to service multiple hyperlinks.

When the container decides to navigate the link as a result of user action it does roughly the following:

```
// retrieve the hyperlink frame pointer
if (!m_fTriedToGetFrame && m_poleinplaceframe && m_phlframe == NULL) {
      m_fTriedToGetFrame = TRUE;
      m_poleinplaceframe->QueryInterface(IID_IHlinkFrame, (void**)&m_phlFrame);
      }

if (m_phlbc == NULL) {
      // get the browse context pointer
      if (m_phlFrame)
            m_phlFrame->GetBrowseContext(&m_phlbc);
      else
            HlinkCreateBrowseContext(&m_phlbc);
      if (m_phlbc == NULL)
            return E_FAIL;

      // register with the browse context
      m_phlbc->Register(pmkThis, phls, &m_dwRegister);

      // because we want the user to be able to come back this object via GoBack functionality add
      // self to the navigation stack. This effectively tells the browse context that the current hyperlink container
      // refered to by pmkThis is a member (at the top) of the navigation stack
      m_phlbc->OnNavigateHlink(NULL, pmkThis, szLocation, szFriendlyName);
      }

      if (m_phlframe == NULL)  {
            HLBWINFO hlbwinfo;

            // initialize hlbwinfo with window locaitons and flags

            // register the browse window info in the browse context so it is later available to the hyperlink target and
            m_phlbc->SetBrowseWindowInfo(&hlbwinfo)

            // set the flag indicating that this window should be hidden after navigation. But this flag
            // is cleared in the IHlinkTarget::Navigate and IHlinkFrame::OnNavigate methods.
            m_fHide = TRUE;
            v_fHideAppFrame = TRUE;  // only the MDI applications need this separate flag
            }
      else {
            m_fHide = FALSE;
            v_fHideAppFrame = FALSE;
      }

      hr = HlinkNavigate(pkl, m_phlframe, NULL,  pbc /* could be NULL */, pibsc, m_phlbc);

      if (hr == NOERROR) {
            if (m_fHide)
```

```
            // Hide this document
        if (m_vfHideAppFrame)
            // Hide/Minimize application's frame window
        }
```

IHlinkFrame::Navigate

The hyperlink frame is called from within HlinkNavigate. This is the frame's chance to provide a integrated user-interface, progress feedback, cancellability, and so forth. Often the hyperlink frame simply sets some flags and defers to IHlink::Navigate.

```
STDMETHODIMP
IHlinkFrame::Navigate(DWORD grfHLNF, IBindCtx* pbc,  IBindStatusCallback* pbsc,  IHlink* phlDest)
{
    // This flag gets cleared in IHlinkFrame::OnNavigate(). Thus if the Navigation within the same
    // frame window then we will get the right behaviour as this flag gets cleared in the OnNavigate()
    // method.
    m_fHide = TRUE;

    // some frames need only the following function so that they can properly show and hide themselves.
    // others may hook themselves in to the IBindStatusCallback for progress notification
    hr = IHlink::Navigate(NULL, pbc, pbsc, phlDest);
    if (SUCCEEDED(hr) && m_fHide)
        // Hide the frame window
} // IHlinkFrame::Navigate
```

Other frames may choose to do much more work. For instance, a frame may decide to provide its own IBindStatusCallback in order to listen in on progress notifications during navigation.

IHlink::Navigate

When called by the hyperlink frame, the system hyperlink object gains control to do the brunt of the navigation work on to integrate the results with the browse context. The code below is a sample implementation of a hyperlink object. The system provided hyperlink object uses similar code:

```
STDMETHODIMP
IHlink::Navigate(DWORD grfHLNF, IBindCtx* pbc,  IBindStatusCallback* pbsc, IHlinkBrowseContext* phlbc)
{
    IHlinkTarget* phlTarget = NULL;
    IMoniker* pmkLeft = NULL;

    if (grfHLNF & HLNF_USEBROWSECONTEXTCLONE) {
        grfHLNF &= ~HLNF_USEBROWSECONTEXTCLONE;
        phlbc->Clone(NULL, IID_IHlinkBrowseContext, &phlbc);
        }
    else {
        hr = m_phlSite->GetMoniker(m_dwSiteData, OLEGETMONIKER_ONLYIFTHERE, OLEWHICHMK_CONTAINER,
        &pmkLeft);
        if (FAILED(hr) || m_pmkTarget->IsEqual(pmkLeft)) {
            hr = m_phlsite->GetInterface(dwSiteData, 0, IID_IHlinkTarget, (void**)&phlTarget);
            if (FAILED(hr))
                phlbc->GetObject(m_pmkTarget, &phlTarget);
            }

    if (phlTarget == NULL) {
        // Set the pbsc in the pbc to get asynch and notification binding behavior requested by caller
        m_pmkTarget->BindToObject(pmkLeft, IID_IHlinkTarget, &phlTarget);
        phlTarget->SetBrowseContext(phlbc);
        }

    phlTarget->Navigate(grfHLNF, m_szLocation);
} // IHlink::Navigate
```

IHlinkTarget::SetBrowseContext

During the execution of the above code, the hyperlink target receives the browse context for the navigation. This provides information about the hyperlink navigation stack and about window positions for the hyperlinking. **Note:** This function is not called in all hyperlinking circumstances, for example in the "simple hyperlinking" example above. A "simple" hyperlink target need not implement this function.

```
STDMETHODIMP
```

```
CHlinkTarget::SetBrowseContext(IHlinkBrowseContext* phlbc)
{
	if (m_phlbc != NULL) {
		m_phlbc->Revoke(m_dwRegister);
		m_phlbc->Release();
		}
	m_phlbc = phlbc;
	if (m_phlbc != NULL) {
		m_phlbc->AddRef();
		m_phlbc->Register(0, (IUnknown*)this, m_pmk, &m_dwRegister);
		}

	return S_OK;
} // CHlinkTarget::SetBrowseContext
```

IHlinkTarget::Navigate

The hyperlink target then receives control to navigate to the specific location within the target.

```
 STDMETHODIMP
IHlinkTarget::Navigate(DWORD grfHLNF, LPCWSTR szLocation)
{
	IHlinkFrame* phlFrame = NULL;

	// if the object is not visible, activate it and show it. jump to the location indicated by szLocation

	// if this hyperlink target is an OLE Document Object, try to retrieve the hyperlink frame pointer from the IOleInPlaceFrame
	if (m_poleinplaceframe)
		m_poleinplaceframe->QueryInterface(IID_IHlinkFrame, (void**)&phlFrame);

	// notify the hlink frame and the browse context that the navigation is complete. Note: either phlFrame or m_phlbc may be NULL
	HlinkOnNavigate(phlFrame, m_phlbc, grfHLNF, m_pmk, szLocation, szFriendlyName);

	if (phlframe == NULL && !(grfHLNF & HLNF_INTERNALJUMP)) {
		HLBWINFO hlbwi;
		phlbc->GetBrowseWindowInfo(&hlbwi);

		// adjust the document and frame windows according to the dimensions in HLBWI
	}

	m_fHide = FALSE;
	m_fHideFrame = FALSE;
} // IHlinkTarget::Navigate
```

IHlinkFrame::OnNavigate

The hyperlink frame next receives notification of a successful navigation from within IHlinkTarget::Navigate in order to reposition its windows and update their visibility. If this is the same frame that hosted the hyperlink container that initiated the navigation, then the m_fHide flag set below will ensure that the frame remains visible.

```
 STDMETHODIMP
IHlinkFrame::OnNavigate(DWORD grfHLNF)
{
	if (!(grfHLNF & HLNF_INTERNALJUMP)) {
		HLBWINFO hlbwi;
		m_phlbc->GetBrowseWindowInfo(&hlbwi);
		// Adjust this document/frame windows according to the dimensions in hlbwi
	}
	m_fHide = FALSE;
} // IHlinkFrame::OnNavigate
```

IHlinkBrowseContext::OnNavigateHlink

Finally, the browse context receives notification of a successful navigation from IHlinkTarget::Navigate in order to update the navigation stack.

```
 STDMETHODIMP
IHlinkBrowseContext::OnNavigateHlink(DWORD grfHLNF, IMoniker* pmkTarget, LPCWSTR szLocation, LPCWSTR
		szFriendlyName)
{
```

```
    // if CreateNoHistory or NavigatingToStackItem, return immediately
    // unless NavigatingBack or NavigatingForward is also set
    if (grfHLNF & (HLNF_CREATENOHISTORY | HLNF_NAVIGATINGTOSTACKITEM)) {
        if (!(grfHLNF & (HLNF_NAVIGATINGBACK | HLNF_NAVIGATINGFORWARD)))
            return NOERROR;
        }
    if (grfHLNF & HLNF_NAVIGATINGBACK)
        --m_iCurrent;
    else if (grfHLNF & HLNF_NAVIGATINGFORWARD)
        ++m_iCurrent;
    else {
        // Add this hyperlink to the navigation stack at m_iCurrent+1, remove all items greater than m_iCurrent+1,

        ++m_iCurrent;
        }
} // IHlinkBrowseContext::OnNavigateHlink
```

Technical Details[12]

```
typedef enum tagHLNF {
    HLNF_INTERNALJUMP,
    HLNF_NAVIGATINGBACK,
    HLNF_NAVIGATINGFORWARD,
    HLNF_USEBROWSECONTEXTCLONE,
    HLNF_OFFSETWINDOWORG,
    HLNF_OPENINNEWWINDOW,
    HLNF_CREATENOHISTORY,
    HLNF_NAVIGATINGTOSTACKITEM,
    } HLNF;

typedef enum {
    HLINKWHICHMK_CONTAINER,
    HLINKWHICHMK_BASE
    } HLINKWHICHMK;

interface IHlinkSite : IUnknown {
    HRESULT     GetMoniker([in] DWORD dwSiteData, [in] DWORD dwAssign, [in] DWORD dwWhich, [out] IMoniker** ppmk);
    HRESULT     GetInterface( [in] DWORD dwSiteData, [in] DWORD dwReserved, [in] REFIID riid, [out, iid_is(riid)] IUnknown** ppv);
    HRESULT     OnNavigationComplete( [in] DWORD dwSiteData, [in] HRESULT hrStatus, [in] LPCWSTR pszStatus);
    };

typedef enum {
    HLINKGETREF_DEFAULT,
    HLINKGETREF_ABSOLUTE,
    HLINKGETREF_RELATIVE
    } HLINKGETREF;

typedef enum {
    HLFNAMEF_DEFAULT,
    HLFNAMEF_TRYCACHE,
    HLFNAMEF_TRYPRETTYTARGET,
    HLFNAMEF_TRYFULLTARGET,
    HLFNAMEF_TRYWIN95SHORTCUT
    } HLFNAMEF;

typedef enum {
    HLINKMISC_ABSOLUTE,
    HLINKMISC_RELATIVE
    } HLINKMISC;

interface IHlink : IUnknown {
    HRESULT     SetHlinkSite([in] IHlinkSite* phlSite, [in] DWORD dwSiteData);
    HRESULT     GetHlinkSite([out] IHlinkSite** pphlSite, [out] DWORD* pdwSiteData);
    HRESULT     GetMonikerReference([in] DWORD dwWhichRef, [out] IMoniker** ppmk, [in, out, unique] LPWSTR* pszLocation);
    HRESULT     GetStringReference([in] DWORD dwWhichRef, [out] LPWSTR* pszTarget, [out] LPWSTR* pszLocation);
    HRESULT     GetFriendlyName([in] DWORD grfHLFNAMEF, [out] LPWSTR* pszFriendlyName);
    HRESULT     SetFriendlyName([in] LPCWSTR szFriendlyName);
    HRESULT     GetTargetFrameName([out] LPWSTR* pszTargetFrameName);
    HRESULT     SetTargetFrameName([in] LPCWSTR szTargetFrameName);
```

[12] For technical details of the "Simple Hyperlink Navigation" API, see the corresponding document, hlsimple.doc.

```
    HRESULT     GetAdditionalParams([out] LPWSTR* pszAdditionalParams);
    HRESULT     SetAdditionalParams([in] LPCWSTR szAdditionalParams);
    HRESULT     Navigate( [in] DWORD grfHLNF, [in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc, [in] IHlinkBrowseContext* phlbc);
    HRESULT     GetMiscStatus([out] DWORD *pdwStatus);
    };

interface IHlinkTarget : IUnknown {
    HRESULT     SetBrowseContext([in, unique] IHlinkBrowseContext* phlbc);
    HRESULT     GetBrowseContext([out] IHlinkBrowseContext** pphlbc);
    HRESULT     Navigate([in] DWORD grfHLNF, [in, unique] LPCWSTR szJumpLocation);
    HRESULT     GetMoniker([in,unique] LPCWSTR szLocation, [in] DWORD dwAssign, [out] IMoniker** ppmkLocation);
    HRESULT     GetFriendlyName([in,unique] LPCWSTR szLocation, [out] LPWSTR* pszFriendlyName);
    };

interface IHlinkFrame : IUnknown {
    HRESULT     SetBrowseContext([in] IHlinkBrowseContext* phlbc);
    HRESULT     GetBrowseContext([out] IHlinkBrowseContext** pphlbc);
    HRESULT     Navigate([in] DWORD grfHLNF, [in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc, [in] IHlink* phlNavigate);
    HRESULT     OnNavigate([in] DWORD grfHLNF);
    };

typedef struct tagHLITEM {
    ULONG       uHLID;
    LPWSTR      szFriendlyName;
    } HLITEM;

typedef Enum<HLITEM*> IEnumHLITEM;

typedef enum tagHLBWIF {
    HLBWIF_HASFRAMEWNDINFO,
    HLBWIF_HASDOCWNDINFO ,
    HLBWIF_FRAMEWNDMAXIMIZED,
    HLBWIF_DOCWNDMAXIMIZED
    } HLBWIF;

typedef struct tagHLBWINFO {
    ULONG       cbSize;
    DWORD       grfHLBWIF;
    RECTL       rcFramePos;
    RECTL       rcDocPos;
    } HLBWINFO;

typedef enum tagHLID {
    HLID_PREVIOUS,
    HLID_NEXT,
    HLID_CURRENT,
    HLID_STACKBOTTOM,
    HLID_STACKTOP
    } HLID;

typedef enum tagHLQF {
    HLQF_ISVALID,
    HLQF_ISCURRENT
    } HLQF;

interface IHlinkBrowseContext : IUnknown {
    HRESULT     Register([in] DWORD dwReserved, [in, unique] IUnknown* punk, [in, unique] IMoniker* pmk, [out] DWORD* pdwRegister);
    HRESULT     GetObject([in, unique] IMoniker* pmk, [out] IUnknown** ppunk);
    HRESULT     Revoke([in] DWORD dwRegister);
    HRESULT     SetBrowseWindowInfo([in, unique] HLBWINFO* phlbwi);
    HRESULT     GetBrowseWindowInfo([out] HLBWINFO* phlbwi);
    HRESULT     EnumNavigationStack([out] IEnumHLITEM** ppenumhlitem);
    HRESULT     QueryHlink([in] DWORD grfHLQF, [in] ULONG uHLID);
    HRESULT     GetHlink([in] ULONG uHLID, [out] IHlink** pphl);
    HRESULT     SetCurrentHlink([in] ULONG uHLID);
    HRESULT     OnNavigateHlink([in] DWORD grfHLNF, [in] IMoniker* pmkTarget, [in] LPCWSTR szLocation, [in] LPCWSTR
        szFriendlyName);
    HRESULT     Clone([in] IUnknown* punkOuter, [in] REFIID riid, [out, iid_is(riid)] IUnknown** ppv);
    HRESULT     Close([in] DWORD dwReserved);
    };
```

```
// CLSID_StdHlink: {79eac9d0-baf9-11ce-8c82-00aa004ba90b}
DEFINE_GUID(CLSID_StdHlink, 0x79eac9d0, 0xbaf9, 0x11ce, 0x8c, 0x82, 0x00, 0xaa, 0x00, 0x4b, 0xa9, 0x0b);

// CLSID_StdHlinkBrowseContext: {79eac9d1-baf9-11ce-8c82-00aa004ba90b}
DEFINE_GUID(CLSID_StdHlinkBrowseContext, 0x79eac9d1, 0xbaf9, 0x11ce, 0x8c, 0x82, 0x00, 0xaa, 0x00, 0x4b, 0xa9,
        0x0b);

#define CFSTR_HYPERLINK             TEXT("HyperLink")
#define CF_HYPERLINK            RegisterClipboardFormat(CFSTR_HYPERLINK)

HRESULT     HlinkCreateBrowseContext([in] IUnknown* punkOuter, [in] REFIID riid, [out, iid_is(riid)] void** ppv);
HRESULT     HlinkQueryCreateFromData([in] IDataObject* pdatobj);
HRESULT     HlinkCreateFromData([in] IDataObject* pdatobj, [in] IHlinkSite* phlSite, [in] DWORD dwSiteData, [in] IUnknown* punkOuter, [in]
        REFIID riid, [out, iid_is(riid)] void** ppv);
HRESULT     HlinkCreateFromMoniker([in] IMoniker* pmkTarget, [in] LPCWSTR szLocation, [in] LPCWSTR szFriendlyName, [in]
        IHlinkSite* phlSite, [in] DWORD dwSiteData, [in] IUnknown* punkOuter, [in] REFIID riid, [out, iid_is(riid)] void** ppv);
HRESULT     HlinkCreateFromString([in] LPCWSTR szTarget, [in] LPCWSTR szLocation, [in] LPCWSTR szFriendlyName, [in] IHlinkSite*
        phlSite, [in] DWORD dwSiteData, [in] IUnknown* punkOuter, [in] REFIID riid, [out, iid_is(riid)] void** ppv);

typedef enum {
    HLSR_HOME,
    HLSR_SEARCHPAGE,
    HLSR_HISTORYFOLDER
    } HLSR;

HRESULT     HlinkGetSpecialReference([in] DWORD dwReference, [out] LPWSTR** pszReference);
HRESULT     HlinkSetSpecialReference([in] DWORD dwReference, [in] LPCWSTR szReference)
HRESULT     HlinkNavigateToStringReference([in] LPCWSTR szTarget, [in] LPCWSTR szLocation, [in] IHlinkSite* phlSite, [in] DWORD
        dwSiteData, [in] IHlinkFrame* phlframe, [in] DWORD grfHLNF, [in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc, [in] IHlinkBrowseContext*
        phlbc);
HRESULT     HlinkNavigate ([in] IHlink *phl, IHlinkFrame* phlFrame, [in] DWORD grfHLNF, [in] IBindCtx* pbc, [in] IBindStatusCallback* pbsc,
        [in] IHlinkBrowseContext* phlbc);
HRESULT     HlinkOnNavigate([in] IHlinkFrame* phlframe, [in] IHlinkBrowseContext* phlbc, [in] DWORD grfHLNF, [in] IMoniker* pmkTarget,
        [in] LPCWSTR szLocation, [in] LPCWSTR szFriendlyName);
```

HLNF Enumeration

Values from the HLNF enumeration are used to indicate how 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.

Value	Description
HLNF_INTERNALJUMP	The navigation is an internal jump within the current hyperlink target. The system-provided Hyperlink object will add 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::GetBrowseWindowContext 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.

HLINKWHICHMK Enumeration

A single value from the HLINKWHICHMK enumeration is passed to IHlinkSite::GetMoniker to specify whether the client is requesting the moniker for the container document or a base moniker specific to the site.

Member	Description
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)

HLINKGETREF Enumeration

A single value from the HLINKGETREF enumeration is passed to the IHlink::GetMonikerReference and IHlink::GetStringReference methods to specify whether the client is requesting the absolute reference for the hyperlink target.

Member	Description
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.

HLFNAMEF Enumeration

A single value from the HLFNAMEF enumeration is passed to IHlink::GetFriendlyName to specify which friendly name the client is requesting.

Member	Description
HLFNAMEF_TRYCACHE	Requests the friendly name that is cached in the Hlink object.
HLFNAMEF_TRYFULLTARGET	Requests the full display name of the hyperlink target.
HLFNAMEF_TRYPRETTYTARGET	Requests a beautified version of the display name of the hlink target.
HLFNAMEF_TRYWIN95SHORTCUT	Requests a simplified version of the full display name of the hyperlink target (i.e. after stripping the path and the extension).
HLFNAMEF_DEFAULT	Requests the cached friendly name, else the simplified display name.

HLINKMISC Enumeration

A single value from the HLINKMISC enumeration is returned from IHlink::GetMiscStatus specifying whether the hyperlink object is a relative or an absolute hyperlink.

Member	Description
HLINKMISC_ABSOLUTE	The given hyperlink object contains an absolute reference to the hyperlink target.
HLINKMSIC_RELATIVE	The given hyperlink object contains a relative reference to the hyperlink target.

HLITEM Structure

This structure is returned from IEnumHLITEM::Next calls on enumerators returned from IHlinkBrowseContext::EnumNavigationStack.

Member	Type	Description
uHLID	ULONG	Identifies the hyperlink. Standard enumerators never return one of the logical HLID constants in this field, always an identifier.
szFriendlyName	LPWSTR	Friendly name of the hyperlink. Appropriate for display in the user interface.

HLBWIF Enumeration

HLBWIF flags are passed as part of the HLBWINFO structure which is associated with each browse context. The HLBWINFO structure is retrieved from the browse context using IHlinkBrowseContext::GetBrowseWindowContext, and put into the browse context using IHlinkBrowseContext::SetBrowseWindowContext.

Value	Description
HLBWIF_HASFRAMEWNDINFO	Indicates that this browse context has available frame-level window positioning information.
HLBWIF_HASDOCWNDINFO	Indicates that 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.

HLBWINFO Structure

Contains information 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::GetBrowseWindowContext, and put into the browse context using IHlinkBrowseContext::SetBrowseWindowContext. 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.

Member	Type	Description
cbSize	ULONG	Total size of this structure in bytes.
grfHLBWIF	DWORD	Values taken from the HLBWIF enumeration.
rcFramePos	RECTL	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	RECTL	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.

HLID Constants

For convenience and performance, individual hyperlink objects are often identified in a navigation stack (the browse context or a history/favorites list) using a ULONG identifier – an 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*. Several HLID values are reserved and identify logical positions within a navigation stack.

Member	Description
HLID_PREVIOUS	Indicates 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	Indicates 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	Indicates 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	Indicates 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	Indicates 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.

HLQF Constants

A single value from the HLQF enumeration is passed to IHlinkBrowseContext::QueryHlink to allow the caller to determine the state of a particular hyperlink.

Member	Description
HLQF_ISVALID	Used to test 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	Used to test if the specific hyperlink (identified by the uHLID parameter) is the user's current position within the navigation stack.

CF_HYPERLINK Clipboard Format

The CF_HYPERLINK format consists of a serialized hyperlink. When occurring as part of Uniform Data Transfer in an IDataObject, the format may appear in either TYMED_ISTREAM or TYMED_HGLOBAL mediums. Any hyperlink that supports IPersistStream can be placed into an IStream using OleSaveToStreamEx

HLSR Enumeration

A single value from the HLSR enumeration is passed to HlinkGetSpecialReference or HlinkSetSpecialReference to determine which value to set or get.

Member	Description
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 reference to the global user “history folder” page.

HlinkCreateBrowseContext

HRESULT HlinkCreateBrowseContext(punkOuter, riid, ppv);

Creates an empty, default instance of the system browse context object. This helper API is identical to calling CoCreateInstance(CLSID_StdHlinkBrowseContext, punkOuter, CLSCTX_SERVER, riid, ppv).

Argument	Type	Description
punkOuter	IUnknown*	Controlling IUnknown for the new browse context. Typically NULL, in which case the new browse context is not aggregated.
fcriid	REFIID	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	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

HlinkQueryCreateFromData

HRESULT HlinkQueryCreateFromData(pdatobj);

Determines if a hyperlink can be created from a given IDataObject. A hyperlink can be created from an IDataObject if either,

1. The IDataObject offers CF_HYPERLINK on either TYMED_ISTREAM or TYMED_HGLOBAL.
2. The IDataObject offers *Win95 shortcut* data.[13]

Argument	Type	Description
pdatobj	IDataObject*	The source data object to query about the availability of hyperlink formats.
Returns	S_OK	Yes, a hyperlink can be created from the data.
	S_FALSE	No, a hyperlink can not be created from the data.
	E_INVALIDARG	One or more arguments are invalid.

HlinkCreateFromData

HRESULT HlinkCreateFromData(pdatobj, phlSite, dwSiteData, punkOuter, riid, ppv);

Creates a standard hyperlink object from an IDataObject. Typically the IDataObject originates from a data transfer operation, such as copy-paste using the clipboard, or via drag-and-drop. In the clipboard transfer case, the application retrieves the IDataObject pointer via OleGetClipboard when processing a paste command. During drag-and-drop, the IDataObject is passed in through IDropTarget::Drop to the IDropTarget registered (using RegisterDragDrop) to the window over which the mouse was released during the drag operation.

Once an application obtains an IDataObject, it enumerates available formats to determine how the new data is to merge with existing data. Typically, applications check first for highest-fidelity formats, such as OLE embedding or link objects and their own native data formats, next for medium-fidelity transfer

[13] Further details about this format or how exactly this works to be determined.

formats, such as CF_RTF, CF_METAFILEPICT, CF_DIB, and so on, and finally for low-fidelity transfer formats, such as CF_TEXT. The exact order of course depends on the context of the paste or drop operation and of course on the application itself and its user interaction model.

The following code would typically be inserted at some point in paste or drop logic to allow for pasting and dropping of hyperlinks:

```
if (HlinkQueryCreateFromData(pdatobj) == S_OK) {
    // create a hyperlink site and other hyperlink-specific information as needed
    hr = HlinkCreateFromData(pdatobj, &hlSiteNew, dwSiteData, NULL, IID_IHlink, (void**)&hlNew);
    }
```

Argument	Type	Description
pdatobj	IDataObject*	The source data to create the hyperlink from.
phlSite	IHlinkSite*	The site for the new hyperlink object.
dwSiteData	DWORD	Additional site data for the new hyperlink object.
punkOuter	IUnknown*	Controlling IUnknown for the new hyperlink object. Typically NULL, in which case the new hyperlink is not aggregated.
riid	REFIID	Identifies the interface to return on the new hyperlink object. Typically IID_IHlink, although it must be IID_IUnknown when punkOuter is non-NULL so that the aggregator can retrieve the new hyperlink's inner IUnknown for future delegation of QueryInterface. See the COM aggregation documentation for details.
ppv	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_NOINTERFACE	The object did not support the riid interface.
	E_INVALIDARG	One or more arguments are invalid.

HlinkCreateFromMoniker

HRESULT HlinkCreateFromMoniker(pmkTarget, szLocation, szFriendlyName, phlSite, dwSiteData, punkOuter, riid, ppv);

Creates a new system hyperlink object from a moniker, a location string, and a friendly name. This function may be significantly faster than HlinkCreateFromString if a target moniker is already in-hand.

This API is typically used by hyperlink containers as part of user-interface which create a new hyperlink based on an existing hyperlink, or which allows editing of an existing hyperlink. The following example demonstrates creating a new hyperlink phlNew from an existing hyperlink, phl, by changing only the location within the hyperlink target.

```
phl->GetMonikerReference(&pmk, &szLocation);
phl->GetFriendlyName(&szFriendlyName);
// present UI allowing the user to change the destination of the hyperlink within the same hyperlink target
// show them szLocation, allow them to change it to szLocationNew, same for szFriendlyName
HlinkCreateFromMoniker(pmk, szLocationNew, szFriendlyNameNew, &hlSite, dwSiteData, NULL, IID_IHlink, &phlNew);
```

Argument	Type	Description
pmkTarget	IMoniker*	The moniker to the hyperlink target for the new hyperlink. May not be NULL.
szLocation	LPCWSTR	The string representing the location within the hyperlink target for the new hyperlink. May not be NULL.
szFriendlyName	LPCWSTR	The string to use as the friendly name for the hyperlink.
phlSite	IHlinkSite*	The site for the new hyperlink object.
dwSiteData	DWORD	Additional site data for the new hyperlink object.
punkOuter	IUnknown*	Controlling IUnknown for the new hyperlink. Typically NULL, in which case the new hyperlink is not aggregated.
riid	REFIID	Identifies the interface to return on the new hyperlink. Typically IID_IHlink, 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	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

HlinkCreateFromString

HRESULT HlinkCreateFromString(szTarget, szLocation, szFriendlyName, phlSite, dwSiteData, punkOuter, riid, ppv);

Creates a new hyperlink object from strings representing the hyperlink target, the location within the target, and a friendly name. This function may take some time, as parsing a target string into a moniker may be as expensive as actually binding to the resulting hyperlink.

This API is typically used by hyperlink containers as part of 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.

Argument	Type	Description
szTarget	LPCWSTR	String which helps identify the hyperlink target. This string is resolved into a moniker via MkParseDisplayNameEx.
szLocation	LPCWSTR	The string representing the location within the hyperlink target for the new hyperlink.
szFriendlyName	LPCWSTR	The string to use as the friendly name for the hyperlink.
phlSite	IHlinkSite*	The site for the new hyperlink object.
dwSiteData	DWORD	Additional site data for the new hyperlink object.
punkOuter	IUnknown*	Controlling IUnknown for the new hyperlink. Typically NULL, in which case the new hyperlink is not aggregated.
riid	REFIID	Identifies the interface to return on the new hyperlink. Typically IID_IHlink, 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	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

HlinkGetSpecialReference

HRESULT HlinkGetSpecialReference(dwReference, pszReference);

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. As an example, applications use this API to retrieve the string (convertible to a hyperlink via HlinkCreateFromString) which they navigate to on launch or when executing a *Go Home* command.

Note: need to make sure everybody is using the same registry locations! <HadiP>

Argument	Type	Description
dwReference	DWORD	A value taken from the HLSR enumeration.
pszReference	LPWSTR*	Location to return the string to the global default page. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	The arguments are invalid.

HlinkSetSpecialReference

HRESULT HlinkSetSpecialReference(dwReference, szReference);

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 an example, applications use this API to implement a *Set Home* command, which sets the currently visible navigation point as the home page for the current user.

Argument	Type	Description
dwReference	DWORD	A value taken from the HLSR enumeration.
szReference	LPCWSTR	The string to set to the global default page.
Returns	S_OK	Success.
	E_INVALIDARG	The arguments are invalid.

HlinkNavigateToStringReference

HRESULT HlinkNavigateToStringReference(szTarget, szLocation, phlSite, dwSiteData, phlFrame, grfHLNF, pbc, pbsc, phlbc);

Simple Helper function which encapsulates the following common sequence of calls:

```
// 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();
```

Argument	Type	Description
szTarget	LPCWSTR	String which helps identify the hyperlink target. This string is resolved into a moniker for underlying binding operations via MkParseDisplayNameEx.
szLocation	LPCWSTR	The string representing the location within the hyperlink target for the new hyperlink.
phlSite	IHlinkSite*	The site for the new hyperlink object. (optional, in which case szTarget must be an absolute reference)
dwSiteData	DWORD	Additional site data for the new hyperlink object.
phlFrame	IHlinkFrame*	The hyperlink frame of the hyperlink container. May be NULL if the hyperlink container does not have a hyperlink frame.
grfHLNF	DWORD	Values taken from the HLNF enumeration.
pbc	IBindCtx*	The bind context to use for any moniker binding performed during the navigation. May not be NULL.
pbsc	IBindStatusCallback*	The 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.
phlbc	IHlinkBrowseContext*	The browse context to use for this navigation. The browse context includes history information in which this navigation is logged, if !(grfHLNF & HLNF_CREATENOHISTORY).
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

HlinkNavigate

HRESULT HlinkNavigate (phl, phlFrame, grfHLNF, pbc, pbsc, phlbc);

This function can be used to navigate a hyperlink given a hyperlink object and an optional hyperlink fram eobject. This helper function which encapsulates the following common sequence of calls:

```
if (phlFrame)
    phlFrame->Navigate(grfHLNF, pbc, pbsc, phl);
else if (phl)
    phl->Navigate(grfHLNF, pbc, pbsc, phlbc);
```

Argument	Type	Description
phl	Ihlink *	The hyperlink.to navigate to.
phlFrame	IHlinkFrame*	The hyperlink frame of the hyperlink container. May be NULL if the hyperlink container does not have a hyperlink frame.
grfHLNF	DWORD	Values taken from the HLNF enumeration.
pbc	IBindCtx*	The bind context to use for any moniker binding performed during the navigation. May not be NULL.
pbsc	IBindStatusCallback*	The 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.
phlbc	IHlinkBrowseContext*	The browse context to use for this navigation. The browse context includes history information in which this navigation is logged, if !(grfHLNF & HLNF_CREATENOHISTORY).
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

HlinkOnNavigate

HRESULT HlinkOnNavigate(phlFrame, phlbc, grfHLNF, pmkTarget, szLocation, szFriendlyName);

Encapsulates a sequence of common steps performed in hyperlink targets during IHlinkTarget::Navigate, namely the calling of IHlinkBrowseContext::OnNavigateHlink and IHlinkFrame::OnNavigate if the hyperlink target has a hyperlink frame. This function encapsulates the following functionality:

```
phlbc->OnNavigateHlink(grfHLNF, pmkTarget, szLocation, szFriendlyName);
if (phlframe)
    phlframe->OnNavigate(grfHLNF);
```

Argument	Type	Description
phlFrame	IHlinkFrame*	The hyperlink frame of the hyperlink container. May be NULL if the hyperlink container does not have a hyperlink frame.
phlbc	IHlinkBrowseContext*	The browse context being used for this navigation. The browse context includes this navigation in its history information during IHlinkBrowseContext::OnNavigateHlink if ! (grfHLNF & HLNF_CREATENOHISTORY).
grfHLNF	DWORD	Values taken from the HLNF enumeration.
pmkTarget	IMoniker*	The moniker of the hyperlink target. May not be NULL.
szLocation	LPCWSTR	The string representing the location within the hyperlink target for the new hyperlink. May not be NULL.
szFriendlyName	LPCWSTR	The friendly name of the hyperlink.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkSite::GetMoniker

HRESULT IHlinkSite::GetMoniker(dwSiteData, dwAssign, dwWhich, ppmk);

Returns the moniker of the hyperlink site's container. Called by a hyperlink on its hyperlink site to retrieve a relative moniker during IHlink::Navigate so that the hyperlink can determine if the navigation is internal (within the same container) or not.

Argument	Type	Description
dwSiteData	DWORD	Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.
dwAssign	DWORD	A value from the OLEGETMONIKER enumeration. Typically OLEGETMONIKER_ONLYIFTHERE, indicating that 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	DWORD	A value from the OLEWHICHMK enumeration. Typically OLEWHICHMK_CONTAINER, indicating that the site should return the moniker of the hyperlink container.
ppmk	IMoniker**	Location to return the IMoniker interface of the specific moniker.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkSite::GetInterface

HRESULT IHlinkSite::GetInterface(dwSiteData, dwReserved, riid, ppv);

Retrieves an interface on the hyperlink container (usually the document that contains the hyperlink site). A hyperlink typically calls this method on its hyperlink site after calling IHlinkSite::GetMoniker and determining that there is no relative moniker as part of IHlink::Navigate. The hyperlink next calls this method with riid==IID_IHlinkTarget to retrieve the hyperlink target of the container so that the hyperlink can directly call IHlinkTarget::Navigate, avoiding the typical moniker binding process.

Note: IHlinkSite::GetInterface behaves similarly to QueryInterface, except that it may choose what interface to return based on the dwSiteData parameter. Furthermore, this interface is not necessarily implemented on the same object that exposes IHlinkSite.

Argument	Type	Description
dwSiteData	DWORD	Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.
dwReserved	DWORD	Reserved for future use. Must be zero.
riid	REFIID	Identifies the interface to return.
ppv	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_NOINTERFACE	The desired interface is not available.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkSite::OnNavigationComplete

HRESULT IHlinkSite::OnNavigationComplete(dwSiteData, hrStatus, pszStatus);

The hyperlink object calls this method on the hyperlink site to notify it that a hyperlink has completed navigation. This notification is particularly useful if the hyperlink has been navigated asynchronously, because it is the only notification the hyperlink receives to realize that hyperlinking has completed.

Argument	Type	Description
dwSiteData	DWORD	Identifies the hyperlink to the hyperlink site. The hyperlink site initializes the hyperlink with this value as part of IHlink::SetHlinkSite.
hrStatus	HRESULT	Result of the hyperlink navigation. Either S_OK for success or E_ABORT or E_FAIL.
pszStatus	LPCWSTR	A string describing the failure that occurred.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink Interface

IHlink::SetHlinkSite

HRESULT IHlink::SetHlinkSite(phlSite, dwSiteData);

Sets the hyperlink site and associated site data on a hyperlink object. A hyperlink container typically constructs a hyperlink site first and then passes it through this method to a newly constructed hyperlink object. The hyperlink object uses the hyperlink site in order to navigate properly when IHlink::Navigate is called. Calls on this method are often encapsulated in helper functions such as HlinkCreateFromData, HlinkCreateFromMoniker, and HlinkCreateFromString.

Argument	Type	Description
phlSite	IHlinkSite*	The new hyperlink site for this hyperlink.
dwSiteData	DWORD	Further site data to be kept on behalf of the site.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetHlinkSite

HRESULT IHlink::GetHlinkSite(pphlSite, pdwSiteData);

Returns the hyperlink site and associated site data from a hyperlink object. This method is infrequently used by clients.

Argument	Type	Description
pphlSite	IHlinkSite**	Location to return the IHlinkSite interface. May not be NULL.
pdwSiteData	DWORD*	Location to return the site data. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetMonikerReference

HRESULT IHlink::GetMonikerReference(dwWhichRef, ppmk, pszLocation);

Returns the moniker and location portions of the hyperlink reference. The moniker portion of the hyperlink reference can bind to the hyperlink target of the reference via IMoniker::BindToObject(..., IID_IHlinkTarget,...). The location portion of the hyperlink reference can be passed to the hyperlink target

via IHlinkTarget::Navigate to navigate to the proper destination within the target, or it can be used to retrieve the current friendly name of the location within the target via IHlinkTarget::GetFriendlyName.

Argument	Type	Description
dwWhichRef	DWORD	Value from the HLINKGETREF enumeration specifying whether to get the absolute or relative reference to the hyperlink target.
ppmk	IMoniker**	Location to return the moniker to the hyperlink target of the hyperlink reference, if any. May be NULL, in which case the caller is not interested in the moniker to the hyperlink target.
pszLocation	LPWSTR*	Location to return the location portion of the hyperlink reference, if any. May be NULL, in which case the caller is not interested in the location portion of the hyperlink reference.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetStringReference

HRESULT IHlink::GetStringReference(dwWhichRef, pszTarget, pszLocation);

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

The pszTarget string is typically retrieved by calling IMoniker::GetDisplayName on the moniker to the hyperlink target that the hyperlink contains within it.

Argument	Type	Description
dwWhichRef	DWORD	Value from the HLINKGETREF enumeration specifying whether to get the absolute or relative reference to the hyperlink target.
pszTarget	LPWSTR*	Location to return a string that helps identify the hyperlink target of the hyperlink reference. May be NULL, in which case the caller is not interested in the target string of the hyperlink reference.
pszLocation	LPWSTR*	Location to return the location portion of the hyperlink reference. May be NULL, in which case the caller is not interested in the location portion of the hyperlink reference.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetFriendlyName

HRESULT IHlink::GetFriendlyName(grfHLFNAMEF, pszFriendlyName);

Retrieves the friendly name of the hyperlink reference. Since the friendly name may be cached as part of the hyperlink object itself, it may therefore not necessarily correspond to the friendly name that the hyperlink target would return were it activated and queried via IHlinkTarget::GetFriendlyName. The system-provided hyperlink implementation does cache the friendly name, and updates it as part of IHlink::Navigate.

This method is typically called by hyperlink containers, which use friendly names to represent hyperlinks within their user interface.

Argument	Type	Description
grfHLFNAMEF	DWORD	Get from Srini.
pszFriendlyName	LPWSTR*	Location to return the friendly name of the hyperlink reference. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::SetFriendlyName

HRESULT IHlink::SetFriendlyName(szFriendlyName);

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

Argument	Type	Description
pszFriendlyName	LPCWSTR	The friendly name of the hyperlink reference.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetTargetFrameName

HRESULT IHlink::GetTargetFrameName(pszTargetFrameName);

Retrieves the name of the target-frame for the hyperlink. This string names the target frame (as in HTML frame-sets) in which the hyperlink navigation is to occur. This string is only useful if the hyperlink is navigated to in a context/container that understands named frame-sets.

Argument	Type	Description
pszTargetFrameName	LPWSTR*	Location to return the target frame name. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::SetTargetFrameName

HRESULT IHlink::SetTargetFrameName(szTargetFrameName);

Sets the target frame name for the hyperlink. This string names the target frame (as in HTML frame-sets) in which the hyperlink navigation is to occur. This string is only useful if the hyperlink is navigated to in a context/container that understands named frame-sets.

Argument	Type	Description
pszTargetFrameName	LPCWSTR	The target frame name for the hyperlink.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetAdditionalParams

HRESULT IHlink::GetAdditionalParams(pszAdditionalParams);

Retrieves additional parameters or properties for the hyperlink. This parameter string is an extensible string in the following format:

<ID$_1$ = “value$_1$” > <ID$_2$ = “value$_2$”> … <Id$_n$ = “value$_n$”>

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

Argument	Type	Description
pszAdditionalParams	LPWSTR*	Location to return the additional parameters of the hyperlink. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::SetAdditionalParams

HRESULT IHlink::SetAdditionalParams(szAdditionalParams);

Sets the additional parameters or properties for the hyperlink. This parameter string is an extensible string in the following format:

<ID_1 = "$value_1$" > <ID_2 = "$value_2$"> … <Id_n = "$value_n$">

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

Argument	Type	Description
pszAdditionalParams	LPCWSTR	The additional parameters for the hyperlink.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::Navigate

HRESULT IHlink::Navigate(grfNLF, pbc, pbsc, phlbc);

.

Argument	Type	Description
grfNLF	DWORD	Values taken from the HLNF enumeration.
pbc	IBindCtx*	The bind context to use for any moniker binding performed during the navigation. May not be NULL.
pbsc	IBindStatusCallback*	The 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.
phlbc	IHlinkBrowseContext*	The browse context to use for this navigation. May not be NULL. As part of navigation, this browse context's navigation stack may be updated (depending on grfHLNF) and its cache of hyperlink targets will be consulted for matching hyperlink targets.
Returns	S_OK	Success.
	HLINK_S_NAVIGATEDTOLEAFNODE	TBD.
	E_INVALIDARG	One or more arguments are invalid.

IHlink::GetMiscStatus

HRESULT IHlink::GetMiscStatus(*pdwStatus);

Queries whether the hyperlink is an absolute or a relative link.

Argument	Type	Description
pdwStatus	DWORD*	Location to return a value from the HLINKMISC enumeration. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkTarget Interface

A hyperlink target implements this interface to allow navigation to locations within its objects. A typical implementation of a hyperlink target keeps track of only a few extra values that keep track of the current hyperlinking browse context.

```
class CHlinkTarget : public IHlinkTarget {
    // prototypes for IHlinkTarget and IUnknown implementation …

    IHlinkBrowseContext*    m_phlbc;
    IHlinkFrame*            m_phlframe;
    IMoniker*               m_pmk;
    DWORD                   m_dwRegister;
    BOOL                    m_fHide;
    BOOL                    m_fHideFrame;
    };
```

IHlinkTarget::SetBrowseContext

HRESULT IHlinkTarget::SetBrowseContext(phlbc);

Establishes the current browse context for this hyperlink target. Since a hyperlink target has only one browse context at a time, a typical implementation of this method is to hold a reference to the new browse context and release any references to prior browse contexts, as shown in the code sample shown earlier in this document.

Besides registering itself with its browse context during this method, a hyperlink target also uses its browse context during to notify the browse context of a navigation event by calling IHlinkBrowseContext::OnNavigateHlink during IHlinkTarget::Navigate.

NOTE: for simple hyperlink targets it is not necessary to implement this method. If you wish to participate as a hyperlink target but you do not wish to integrate with the system browse context, you may return E_NOTIMPL from this call.

Argument	Type	Description
phlbc	IHlinkBrowseContext*	The browse context to set for the hyperlink target.
Returns	S_OK	Success.
	E_NOTIMPL	This hyperlink target does not understand browse contexts.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkTarget::GetBrowseContext

HRESULT IHlinkTarget::GetBrowseContext(pphlbc);

Retrieves the browse context which this hyperlink target is currently running within. The following code example demonstrates the implementation of this method by a typical hyperlink target holding a reference to its browse context:

```
 STDMETHODIMP
CHlinkTarget::GetBrowseContext(IHlinkBrowseContext** pphlbc)
{
    *pphlbc = m_phlbc;
    if (m_phlbc)
        m_phlbc->AddRef();
    return S_OK;
}  // CHlinkTarget::GetBrowseContext
```

NOTE: for simple hyperlink targets it is not necessary to implement this method. If you wish to participate as a hyperlink target but you do not wish to integrate with the system browse context, you may return E_NOTIMPL from this call.

Argument	Type	Description
pphlbc	IHlinkBrowseContext*	Location to return the IHlinkBrowseContext interface of the current browse context.
Returns	S_OK	Success.
	E_NOTIMPL	This hyperlink target does not understand browse contexts.
	E_INVALIDARG	The pphlbc argument is invalid.

IHlinkTarget::Navigate

HRESULT IHlinkTarget::Navigate(grfHLNF, szLocation);

If the given location is not visible, navigates to and shows the szLocation position within the object/document. The code example earlier in this document demonstrates an implementation of this method.

NOTE: for simple hyperlink targets that do not integrate with the system browse context, it is not necessary to remember and use the browse context and window position info for this operation.

Argument	Type	Description
grfHLNF	DWORD	Values taken from the HLNF enumeration.
szLocation	LPCWSTR	Location within the hyperlink target to navigate to.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkTarget::GetMoniker

HRESULT IHlinkTarget::GetMoniker(szLocation, dwAssign, ppmk);

Returns a moniker to the hyperlink target object for the given hyperlink destination, szLocation. Most hyperlink targets return the same moniker for every szLocation, however this is not required, since different monikers may bind to the same IHlinkTarget instance yet yield different contextual information to the underlying object. A simple hyperlink target would have an implementation such as the sample below:

```
STDMETHODIMP
CHlinkTarget::GetMoniker(LPCWSTR szLocation, DWORD dwAssign, IMoniker** ppmk)
{
    // possibly create moniker dynamically according to dwAssign, simple targets will keep one on hand
    *ppmk = m_pmk;
    if (m_pmk)
        m_pmk->AddRef();
    return S_OK;
} // CHlinkTarget::GetMoniker
```

Argument	Type	Description
szLocation	LPCWSTR	Identifies the hyperlink destination within this target.
dwAssign	DWORD	A value from the OLEGETMONIKER enumeration. Must be either OLEGETMONIKER_ONLYIFTHERE, indicating that 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.
ppmk	IMoniker**	Location to return an IMoniker interface.
Returns	S_OK	Success.
	E_FAIL	A moniker does not exist for this hyperlink target and OLEGETMONIKER_ONLYIFTHERE was specified for dwAssign.
	E_INVALIDARG	One or more arguments are invalid.
	others	From moniker creation APIs such as CreateFileMoniker, MkParseDisplayName, etc.

IHlinkTarget::GetFriendlyName

HRESULT IHlinkTarget::GetFriendlyName(szLocation, pszFriendlyName);

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

Argument	Type	Description
szLocation	LPCWSTR	Identifies the hyperlink destination within this target.
pszFriendlyName	LPWSTR*	Location to return the friendly name. This string must be allocated using CoTaskMemAlloc. It is the caller's responsibility to free this string using CoTaskMemFree.
Returns	S_OK	Success.
	E_OUTOFMEMORY	Insufficient memory to return the friendly name.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkFrame Interface

IHlinkFrame::SetBrowseContext

HRESULT IHlinkFrame::SetBrowseContext(phlbc);

Sets the browse context of the hyperlink frame.

Note: for hyperlink frames that do not integrate with the system browse context, it is acceptable to return E_NOTIMPL from this call.

Argument	Type	Description
phlbc	IHlinkBrowseContext*	The browse context to set for the hyperlink frame.
Returns	S_OK	Success.
	E_NOTIMPL	This hyperlink target does not understand browse contexts.
	E_INVALIDARG	The phlbc argument is invalid.

IHlinkFrame::GetBrowseContext

HRESULT IHlinkFrame::GetBrowseContext(pphlbc);

Returns the browse context of the hyperlink frame.

Note: for hyperlink frames that do not integrate with the system browse context, it is acceptable to return E_NOTIMPL from this call.

Argument	Type	Description
pphlbc	IHlinkBrowseContext**	Location to return the browse context of the hyperlink frame.
Returns	S_OK	Success.
	E_NOTIMPL	This hyperlink target does not understand browse contexts.
	E_INVALIDARG	The pphlbc argument is invalid.

IHlinkFrame::Navigate

HRESULT IHlinkFrame::Navigate(grfHLNF, pbc, pbsc, phlNavigate);

Navigates to phlNavigate. Called by a hyperlink object during IHlink::Navigate when phlframe is non-NULL, to allow the hyperlink frame to interpose itself in the navigation process.

Argument	Type	Description
grfHLNF	DWORD	Values taken from the HLNF enumeration.
pbc	IBindCtx*	The bind context to use for any moniker binding performed during the navigation. May not be NULL.
pbsc	IBindStatusCallback*	The 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.
phlNavigate	IHlink*	The hyperlink to navigate to.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.
	others	From IHLink::Navigate.

IHlinkFrame::OnNavigate

HRESULT IHlinkFrame::OnNavigate(grfHLNF);

Notifies the hyperlink frame that a hyperlink has been navigated to. This allows the hyperlink frame to update user interface elements associated with navigation. A hyperlink target calls this method on its hyperlink frame (if any) during IHlinkTarget::Navigate, usually via the helper function HlinkOnNavigate.

Argument	Type	Description
grfHLNF	DWORD	Values taken from the HLNF enumeration.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IEnumHLITEM Interface

The IEnumHLITEM interface is a standard OLE enumeration interface for HLITEM structures. This interface is returned by IHlinkBrowseContext::EnumNavigationStack.

IEnumHLITEM::Next

HRESULT IEnumHLITEM::Next(celt, rgelt, pceltFetched);

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.

Argument	Type	Description
celt	ULONG	The number of elements to retrieve.
rgelt	HLITEM*	Location to return at most celt HLITEM structures. May not be NULL.
pceltFetched	ULONG*	Location to return the actual number of elements returned in rgelt. May be NULL, in which case the caller is not interested in the actual number of elements returned.
Returns	S_OK	Success. celt elements were returned in rgelt.
	S_FALSE	Success, fewer than celt and possibly zero elements were returned in rgelt. The actual number of elements returned is returned through pceltFetched, if non-NULL.
	E_INVALIDARG	One or more arguments are invalid.

IEnumHLITEM::Skip

HRESULT IEnumHLITEM::Skip(celt);

Skips over the next celt elements in the enumeration sequence.

Argument	Type	Description
celt	ULONG	The number of elements which are to be skipped.
Returns	S_OK	Success. celt elements were skipped.
	S_FALSE	Success. Fewer than celt and possibly zero elements were skipped. This indicates that the enumerator has reached the end of the collection.

IEnumHLITEM::Reset

HRESULT IEnumHLITEM::Reset();

Resets the enumeration sequence to the beginning. There is no guarantee that the same set of HLITEMs 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.

Argument	Type	Description
Returns	S_OK	Success.

IEnumHLITEM::Clone

HRESULT IEnumHLITEM::Clone(ppenumhlitem);

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

Argument	Type	Description
ppenumhlitem	IEnumHLITEM**	Location to return the new enumerator. Must be cleared to NULL on failure.
Returns	S_OK	Success.
	E_OUTOFMEMORY	Insufficient memory to create the new enumerator.
	E_INVALIDARG	The ppenumhlitem argument is invalid.

IHlinkBrowseContext Interface

IHlinkBrowseContext::Register

HRESULT IHlinkBrowseContext::Register(dwReserved, punk, pmk, pdwRegister);

Registers an object with the browse context. The browse context maintains a table of moniker-object bindings to facilitate reuse of hyperlink targets during navigation. When a hyperlink navigates to a hyperlink target, this table is consulted (via IHlinkBrowseContext::GetObject) to see if the hyperlink target is already registered and running, thus avoiding launching a new instance of the hyperlink target application and reloading the same document/object.

Currently, only hyperlink targets are required to register themselves the browse context when they receive IHlinkTarget::SetBrowseContext. Hyperlink targets should keep the returned *pdwRegister and unregister themselves from the browse context on shutdown or if their browse context changes (see IHlinkTarget::SetBrowseContext for details).

Argument	Type	Description
dwReserved	DWORD	Reserved for future use. Must be zero.
punk	IUnknown*	The object being registered.
pmk	IMoniker*	Moniker that identifies the object being registered.
pdwRegister	DWORD*	Location to return a value identifying the registration which can be used to subsequently revoke the registration.
Returns	S_OK	Success.
	MK_S_MONIKERALREADYREGISTERED	Indicates that the object was successfully registered, but that 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.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::GetObject

HRESULT IHlinkBrowseContext::GetObject(pmk, ppunk);

Retrieves an object previously registered in the browse context under the name pmk.

Argument	Type	Description
pmk	IMoniker*	Identifies the object being retrieved.
ppunk	IUnknown**	Location to return the IUnknown interface of the object being retrieved.
Returns	S_OK	Success.
	S_FALSE	There was no object registered under pmk in the browse context.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::Revoke

HRESULT IHlinkBrowseContext::Revoke(dwRegister);

Revokes an object previously registered with this browse context using IHlinkBrowseContext::Register.

Argument	Type	Description
dwRegister	DWORD	Identifies the object to revoke. Returned by a previous call to IHlinkBrowseContext::Register.
Returns	S_OK	Success.
	E_INVALIDARG	The dwRegister argument is invalid.

IHlinkBrowseContext::SetBrowseWindowInfo

HRESULT IHlinkBrowseContext::SetBrowseWindowInfo(phlbwi);

Establishes the HLBWINFO structure of this browse context. The HLBWINFO structure contains information about the position and properties of the document- and frame-level windows of other hyperlink frames and documents in the same browse context. This method is called by hyperlink targets and containers whenever their document-level (and optionally their frame-level) user interface is resized – for example under Windows if the user moves their document window, tiles several frame-level applications, or moves the task-bar and causes their windows to receive WM_SIZE, WM_MOVE, or WM_WINDOWPOSCHANGED messages.

Argument	Type	Description
phlbwi	HLBWINFO*	Points to the new HLBWINFO structure for this browse context.
Returns	S_OK	Success.
	E_INVALIDARG	The phlbwi argument is invalid.

IHlinkBrowseContext::GetBrowseWindowInfo

HRESULT IHlinkBrowseContext:: GetBrowseWindowInfo(phlbwi);

Retrieves the HLBWINFO structure currently associated with the browse context. This structure contains information about the position and properties of the document- and frame-level windows of other hyperlink frames and documents in the same browse context. This method is typically called by a hyperlink target during IHlinkTarget::Navigate as part of determining how to position its document-level (and optionally its frame-level) user interface such that the user's experience navigating between disparate applications will be as visually seamless as possible.

Argument	Type	Description
phlbwi	HLBWINFO*	Location to return the HLBWINFO structure.
Returns	S_OK	Success.
	E_INVALIDARG	The phlbwi argument is invalid.

IHlinkBrowseContext::EnumNavigationStack

HRESULT IHlinkBrowseContext:: EnumNavigationStack(ppenumhlitem);

Returns an enumerator which can be used to enumerate the current contents of the navigation stack. The enumerator returns HLITEM structures, which contain references to the underlying hyperlinks (by HLID), and "friendly names" which can be displayed in the user interface. This method is typically called by hyperlink frame objects as part of presenting drop-down lists and dialog boxes with browse-level history lists.

Argument	Type	Description
ppenumhlitem	IEnumHLITEM**	Location to return the IEnumHLITEM enumeration interface over the set of hyperlinks in this navigation stack.
Returns	S_OK	Success.
	E_INVALIDARG	The ppenumhlitem argument is invalid.

IHlinkBrowseContext::QueryHlink

HRESULT IHlinkBrowseContext:: QueryHlink(grfHLQF, uHLID);

Tests the validity of an uHLID value. This method is typically called by hyperlink frame-level user interface elements to determine whether or not to enable commands such as *Go Forward* and *Go Back* by passing HLID_NEXT and HLID_PREVIOUS, for example:

```
// tests if Go Forward should be enabled
phlbc->QueryHlink(HLQF_ISVALID, HLID_NEXT);
// tests if Go Back should be enabled
phlbc->QueryHlink(HLQF_ISVALID, HLID_PREVIOUS);
```

Argument	Type	Description
grfHLQF	DWORD	A single value taken from the HLQF enumeration.
uHLID	ULONG	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.
Returns	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
	E_INVALIDARG	The grfHLQF flags are invalid. grfHLQF must specify either HLQF_ISVALID or HLQF_ISCURRENT.

IHlinkBrowseContext::GetHlink

HRESULT IHlinkBrowseContext::GetHlink(uHLID, pphl);

Retrieves a hyperlink from this browse context.

Argument	Type	Description
uHLID	ULONG	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	IHlink**	Location to return the IHlink interface of the hyperlink.
Returns	S_OK	Success.
	E_FAIL	The specified hyperlink does not exist.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::SetCurrentHlink

HRESULT IHlinkBrowseContext:: SetCurrentHlink(uHLID);

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

Argument	Type	Description
uHLID	ULONG	Identifies the hyperlink to set. May be a value taken from the HLID constants to indicate a logically identified hyperlink, such as HLID_PREVIOUS or HLID_NEXT.
Returns	S_OK	Success.
	E_FAIL	The specified hyperlink does not exist.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::OnNavigateHlink

HRESULT IHlinkBrowseContext::OnNavigateHlink(grfHLNF, pmkTarget, szLocation, szFriendlyName);

Notifies a browse context that a hyperlink has been navigated. Called by the hyperlink target during IHlinkTarget::Navigate to indicate that the hyperlink has been successfully navigated to.

Argument	Type	Description
grfHLNF	DWORD	Values taken from the HLNF enumeration.
pmkTarget	IMoniker*	The moniker of the hyperlink target.
szLocation	LPCWSTR	A string identifying the location within the hyperlink target that was navigated to. May not be NULL.
szFriendlyName	LPCWSTR	The friendly name of the location within the hyperlink target that has been navigated to. May not be NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::Clone

HRESULT IHlinkBrowseContext:: Clone(punkOuter, riid, ppv);

Duplicates a browse context.

Argument	Type	Description
punkOuter	IUnknown*	Controlling IUnknown for the new browse context. Typically NULL, in which case the new browse context is not aggregated.
riid	REFIID	Identifies the interface to return on the new browse context. Typically IID_IHlink, 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	void**	Location to return the riid interface.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

IHlinkBrowseContext::Close

HRESULT IHlinkBrowseContext::Close(dwReserved);

Closes the hyperlink browse context. Releases all hyperlink targets that have been registered with the browse context via IHlinkBrowseContext::Register.

Argument	Type	Description
dwReserved	DWORD	Reserved for future use. Must be zero.
Returns	S_OK	Success.
	E_INVALIDARG	The dwReserved argument is invalid.