Sweeper Programmability Overview

December 1995

Distribution: Public

DRAFT

This document provides a high level description of the OLE Automation interfaces supported in Sweeper.

Introduction

This document provides an complete overview of the Sweeper Programmability Model; a set of OLE Automation compatible[1] objects that allow Open Scripting script writers, OLE Controls, web crawlers, and just about anything else you can think of to easily automate their access to the Internet. There are several components to the overall sweeper programmability interface, each of which is discussed in a separate document in complete detail. The components (and their related documents) are:

Component Description	Specification Filename
The HTML Object. This component provides an OA compatible set of interfaces for navigating *through* an HTML document. This object is implemented on the same object as the HTML DocObject.	HTML Programmability Model.doc
The IExplorer Frame Object. This component is the IExplorer 3.0 frame, which hosts the HTML DocObject as well as other DocObjects. The programmability interface for this frame allows a script writer to execute methods like Forward, Back, as well as to access the History and Favorites lists.	IExplorer Frame Programmability Model.doc
The Win96 Shell Folder Object. Conceptually this is the same thing as the IExplorer frame, except that this is the Win96 shell code.	Win96 Shell Folder Object.doc (does not exist yet)

In addition, the following documents are relevant to this model:

Component Description	Specification Filename
HTML Extensions for COM Objects. In order to host OLE Controls and other COM objects in HTML, an extension to the HTML language is required. This document specifies such an extension (the "EMBED" tag).	HTML Extensions for COM Objects.doc

This document, and it's accompanying documents, uses the term programmability model to describe the sets of objects described here, including the interfaces, methods, and properties found on those objects. The term "object model" has been used in the past (by Office) to describe the same thing (e.g. the Excel Object Model).

All interfaces defined in the Sweeper Programmability Model are "dual"[2]. This means that they inherit from IDispatch and follow some simple rules and thus can be used by late-bound OLE Automation controllers (via IDispatch::GetIdsOfNames and IDispatch::Invoke) as well as by clients that "early-bind" (e.g. use a vtable). Early-bound clients well not only get significantly better performance, but will also be able to enjoy the simplicity and features of the COM programming model (versus having to deal with calling through IDispatch). We use the naming convention of prefixing the name of the interface with "D" (as opposed to "I") to indicate that the interface is a dual interface.

Issues, Ideas, and Notes

- *Controls in HTML should be able to access the HTML object via the IOleClientSite::GetContainer. This gives an Internet aware control the opportunity to have full access, programmatically, to it's containing document..*

Scenarios

The design of the Sweeper programmability model is driven by a set of usage scenarios. These scenarios are outlined below.

[1] It is critical to understand that the importance of OLE Automation is the ability for high-level languages to call on functionality of components in a manner that is natural (read easy for) the high-level language programmer. Originally OLE Automation achieved this through the late binding mechanism of IDispatch::Invoke, but with the invent of "dual" interfaces high-level programming languages, such as VB 4.0, can access objects via good 'ole (pun intended) COM interfaces, while preserving all the benefits those "VB Programmers" are used to. Hence this proposal defines all interfaces as dual.

[2] See page 707 of Inside OLE Second Edition or the Win32® SDK documentation for details of "dual" interfaces.

Scenario 1: Client Side Web Crawler

In this scenario an ISV wants to write a Web Crawler application that runs on the client site, and uses the programmability interface to drive navigate across the web and search for/collect information. The crawler would start with a list of "staring URLs" and recursively burrow into the net, using the links found on pages it encounters to further the search. For this to work the crawler has to be able to

- Search an HTML page for specific text
- Search an HTML page for hyper links
- Specify a URL to jump to

Example Script

The script below demonstrates, naively, how this would work. Note that in reality, the example below would probably be much more complex (for example it would probably want to restrict the recursion to URLs that are in the same domain as the start URL).

```
Dim foundList As Collection
Dim searchedList As Collection
Sub StartCrawl
	' In this example we just have one starting point, but there could be a list
	Set foundList = New Collection
	Set searhcedList = New Collection
	Crawl("http://www.microsoft.com")
End

Sub Crawl(url As String)

	' Have we already found this URL
	Set found = foundList.Item(url)
	If IsObject(found) Then Goto TheEnd:

	' Have we already searched this URL
	Set found = searchedList.Item(url)
	If IsObject(found) Then Goto TheEnd:

	Set htmlObj = GetObject(url)	' Assumes VB's GetObject is updated to take a URL
	searchedList.Add(url,url)
	If htmlObj.Find("Component Object Model") Then
		foundList.Add(url,url)
	End If

	' Recurse links found on this page (depth first: the stuipd way, but it gets the point across)
	For Each link In htmlObj.HyperLinks
		Crawl(link.URL)
	Next Link

TheEnd:
End Sub
```

Scenario 2: Object Interacting with Its Host Page

In this scenario, an OLE Control, or other COM object that is part of the content of a page wishes to interact with the rest of the contents of the page. One reason for doing this would be to have a control that "read" (as in voice synthesis) the contents of the page to the user.

Another, more realistic example would be a "combo box" control that automatically filled itself with the list of H1 (top level headings) tags and allowed the user to select a chapter. This control would be placed in a non-scrolling region of the page (maybe even on a frame). When the user selected an item in the combo box, the control would tell the browser to scroll the selected heading to the top.

The requirements raised by this scenario include:

- Browsers need to provide embedded objects access to the HTML object. This probably means the browser supports an automation interface on it's container object (accessible to the object via IOleClientSite::GetContainer)

- The HTML programming model must allow specific tag types to be enumerated out of a page.
- The programming model must allow clients to force the browser to scroll (and select).

If the object was written in Visual Basic®, the code that filled the combo box might look like this:

```
Set H1Tags = htmlObj.EnumTag("H1")
For Each H1 in H1Tags
    comboBox.AddObject(H1)  ' Assume the combo box in question supports this
Next H1
```

Once the user selects an item in the combo box, the object needs to jump to the correct tag:

```
Sub OnComboSelChange
    htmlObj.ScrollToTop(comboBox.GetCurSel)
End Sub
```

Note that this example is also valid for scripts embedded in a page. That is, instead of having the combo box do the work, a script on the page would do it… The requirements are the same, however.

Scenario 3: Automated Information Retrieval

Scenario 4: Browser Automation

◆ *This applies to the interaction between the DocObject and the frame.*

Script writers want to be able to write scripts that automate the browser frame. Such a script would want to "go forward", "go back", size the window, show/remove the toolbar, and so forth.

One scenario is a script embedded in an HTML document that provides a button to the user saying "This page looks best when the browser is maximized. Press here to maximize the browser window." Rude, but somewhat interesting.

Here's an example that might be found in a VB 4.0 application. In this scenario, the application automates the navigation of the browser for the user, replaying all the links the user visited, scrolling through each page, and pausing before jumping to the next page.

```
Dim ie As DIExplorer
Sub StartIExplore_Click
    Set ie = New Microsoft.IExplorer.1                  ' When the user clicks on the StartIExplore button
    ie.Open URL := "http://www.wsj.com"                 ' IExplorer starts, and he can browse around
End Sub

Sub DoSlideShow_Click                                   ' Later, after the user's navigated around a bit
    For Each hist In ie.History                         ' he can click the "DoSlideShow" button and replay
        ie.Navigate HLink := hist  ' where he's been
        Wait 1000
    Next hist
End Sub

Sub Next_Click
    ie.GoNext
End Sub

Sub Back_Click
    ie.GoPrevious
End Sub

Sub Home_Click
    ie.GoHome
End Sub
```

Scenario 5: Integration with Non-Internet Applications

Applications that want to support the Internet, but cannot afford to implement the full suite of Internet integration technologies (hyperlinks, URL monikers, etc.) would like to be able to simply launch the Internet Explorer and give it a URL to start with. The user can then navigate around the Internet. When the

user is done he closes IExplorer. Or he can "Go Back" (as many times as he jumped). Either way he ends up in the app he started in. People do this today with Netscape's OLE Automation and DDE interfaces.

Design Overview

- History Object? Should be able to get at history at any level.
- Access to Browse Context? Why?

Below is the Sweeper ODL file. The contents of the box below is made up of content linked from all the other documents. This ODL file has been tested with MkTypLib and correctly generates a Type Library.

```
//================================================
// Sweeper Type Library Discription
// Copyright (c) 1995 Microsoft Corporation
// All Rights Reserved.
//================================================
[
	uuid(0002DF00-0000-0000-C000-000000000046),
	version(1.0),
	helpstring("Microsoft Internet Programmabilty Model Object Library")
]
library MSInternetLib
{
importlib("stdole32.tlb");
importlib("OLEPRO32.DLL");

//================================================
// IExplorer Frame Programmabilty
//================================================
	// ---------------------------- HyperLink Object  ----------------------------
	// IID_DHyperLink: {0002DF07-0000-0000-C000-000000000046}
	[
		uuid(0002DF07-0000-0000-C000-000000000046),
		helpstring("HyperLink Object."),
		oleautomation,
		hidden,
		dual
	]
	interface DHyperLink : IDispatch
	{
		// id(0) indicates that this is the "value" member.
		[id(0),propget, helpstring("Returns or sets the Friendly Name for the HyperLink."), helpcontext(0x0000)]
			HRESULT FriendlyName([out, retval] BSTR* pbstrName);
		[id(0),propset, helpstring("Returns or sets the Friendly Name for the HyperLink."), helpcontext(0x0000)]
			HRESULT FriendlyName([in] BSTR bstrName);
		[propget, helpstring("Returns or sets the string reference for the HyperLink."), helpcontext(0x0000)]
			HRESULT Source([out,retval] BSTR* pbstrSource);
		[propset, helpstring("Returns or sets the string reference for the HyperLink."), helpcontext(0x0000)]
			HRESULT Source([in] BSTR bstrSource);
		[helpstring("Jumps to the hyperlink."), helpcontext(0x0000)]
			HRESULT Navigate([in,optional]VARIANT* OpenInNewWindow, [in,optional] VARIANT* NoHistory);
		[propget, helpstring("Returns a pointer to creator of the object."), helpcontext(0x0000)]
			HRESULT Application([out,retval] IDispatch** ppDisp);
		[propget, helpstring("Returns a pointer to the IExplorer Object."), helpcontext(0x0000)]
			HRESULT Parent([out,retval] IDispatch** ppDisp);
	};

	// ---------------------------- History Object  ----------------------------
	typedef
	[
		uuid(0002DF08-0000-0000-C000-000000000046),
		helpstring("Constants for DInternetHistory")
	]
	enum HlinkIDConstants   {
		[helpstring("Previous Item")]	hlidPrevious = 0,
		[helpstring("Next Item")]	hlidNext = 0xFFFFFFFF,
```

```
        [helpstring("Current Item")]    hlidCurrent = 0xFFFFFFFE,
        [helpstring("Last Item")]       hlidStackBottom = 0xFFFFFFFD,
        [helpstring("First Item")]      hlidStackTop = 0xFFFFFFFC
} HlinkIDConstants;

// IID_DInternetHistory: {0002DF04-0000-0000-C000-000000000046}
[
        uuid(0002DF04-0000-0000-C000-000000000046),
        helpstring("Internet History Object."),
        helpcontext(0x0000),
        oleautomation,
        hidden,
        dual
]
interface DInternetHistory : IDispatch
{
        // id(0) indicates that this is the "value" member.
        [id(0), helpstring("Returns a specific Hyperlink object either by HLID or name."), helpcontext(0x0000)]
                HRESULT Item([in] VARIANT* Index, [out, retval] VARIANT* pVarResult);
        [helpstring("Adds a Hyperlink to the collection"), helpcontext(0x0000)]
                HRESULT Add(
                [in] DHyperLink* HLink,
                        [in, optional] VARIANT* Key,
                [in, optional] VARIANT* Before,
                [in, optional] VARIANT* After);
        [helpstring("Returns the number of Hyperlinks in the collection"), helpcontext(0x0000)]
                HRESULT Count([out,retval] long* plCount);
        [helpstring("Removes a Hyperlink from a Collection object"), helpcontext(0x0000)]
                HRESULT Remove([in] VARIANT* Index);
        [id(DISPID_NEWENUM), restricted, propget]
                HRESULT _NewEnum([out, retval] IUnknown** ppUnk);
        [propget, helpstring("Returns a pointer to the IExplorer Object."), helpcontext(0x0000)]
                HRESULT Application([out,retval] IDispatch** ppDisp);
        [propget, helpstring("Returns a pointer to creator of the object."), helpcontext(0x0000)]
                HRESULT Parent([out,retval] IDispatch** ppDisp);
};

// CLSID_StdHlinkHistory
[
        uuid(0002DF02-0000-0000-C000-000000000046),
        helpstring("Internet History Object")
]
coclass HlinkHistory
{
        [default] interface DInternetHistory;
};

// ---------------------------- IExplorer Frame Object  ----------------------------
// IID_DExplorer: {0002DF05-0000-0000-C000-000000000046}
[
        uuid(0002DF05-0000-0000-C000-000000000046),
        helpstring("IExplorer Frame Object."),
        helpcontext(0x0000),
        hidden,
        oleautomation,
        dual
]
interface DExplorer : IDispatch
{
        // Standard OLE Automation required methods and properties
        // id(0) indicates that this is the "value" member.
        [id(0), propget, helpstring("Returns name of the application."), helpcontext(0x0000)]
                HRESULT Name([out,retval] BSTR* pbstrName);
        [propget, helpstring("Returns the full pathname to the IExplorer executable."), helpcontext(0x0000)]
                HRESULT FullName([out,retval] BSTR* pbstrFullName);
        [propget, helpstring("Returns a pointer to the IExplorer Object."), helpcontext(0x0000)]
                HRESULT Application([out,retval] IDispatch** ppDisp);
        [propget, helpstring("Returns a pointer to the IExplorer Object."), helpcontext(0x0000)]
                HRESULT Parent([out,retval] IDispatch** ppDisp);
```

```
        [propget, helpstring("Determines whether IExplorer is visible or hidden."), helpcontext(0x0000)]
            HRESULT Visible([out, retval] boolean* pBool);
        [propput, helpstring("Determines whether IExplorer is visible or hidden."), helpcontext(0x0000)]
            HRESULT Visible([in] boolean Value);
        [propget, helpstring("Returns the active Document."), helpcontext(0x0000)]
            HRESULT Document([out,retval] IDispatch** ppDisp);
        [helpstring("Exits IExplorer and closes the open document."), helpcontext(0x0000)]
            HRESULT Quit();

        // IExplorer specific methods and properties
        [helpstring("Opens a file."), helpcontext(0x0000)]
            HRESULT Open( [in] BSTR Source);
        [helpstring("Prints the current document."), helpcontext(0x0000)]
            HRESULT PrintOut([in] long What, [in] VARIANT Numbering, [in, optional] VARIANT FirstPage, [in,
        optional]       VARIANT Sections, [in, optional] VARIANT FileName, [in, optional] VARIANT PrinterName,
        [in, optional]          VARIANT DriverName, [in, optional] VARIANT NoPrinting);
        [helpstring("Navigates to a hyperlink."), helpcontext(0x0000)]
            HRESULT Navigate([in]DHyperLink* Hlink,
                             [in,optional]VARIANT* OpenInNewWindow,
                             [in,optional] VARIANT* NoHistory);
        [helpstring("Navigates to the previous item in the history list."), helpcontext(0x0000)]
            HRESULT GoBack();
        [helpstring("Navigates to the next item in the history list."), helpcontext(0x0000)]
            HRESULT GoForward();
        [helpstring("Go home/start page."), helpcontext(0x0000)]
            HRESULT GoHome();
        [helpstring("Stops opening a file."), helpcontext(0x0000)]
            HRESULT Stop();
        [helpstring("Refreshes the current file."), helpcontext(0x0000)]
            HRESULT Refresh();
        [propget, helpstring("Returns the history list."), helpcontext(0x0000)]
            HRESULT History([out, retval] DInternetHistory** ppDInternetHistory);
        [propget, helpstring("Returns the favorites list."), helpcontext(0x0000)]
            HRESULT Favorites([out, retval] DInternetHistory** ppDInternetHistory);
        [propget, helpstring("Returns the the settings object."), helpcontext(0x0000)]
            HRESULT Settings([out, retval] DInternetSettings** ppDInternetSettings);
    };

    // IID_ExplorerEvents: {0002DF06-0000-0000-C000-000000000046}
    [
        uuid(0002DF06-0000-0000-C000-000000000046),
        helpstring("IExplorer Frame Events."),
        helpcontext(0x0000),
        hidden,
        oleautomation,
        dual
    ]
    interface ExplorerEvents : IDispatch
    {
        [helpstring("Allows the recipient to cancel the download."), helpcontext(0x0000)]
            HRESULT Downloading([in,out]OLE_CANCELBOOL* Cancel);
    };

    // CLSID_IExplorer
    [
        uuid(0002DF01-0000-0000-C000-000000000046),
        helpstring("IExplorer Object")
    ]
    coclass IExplorer
    {
        [default] interface DExplorer;
    [default, source] interface ExplorerEvents;
    };


};
```

Appendix A: Assigned GUIDs

The following range of 256 GUIDs has been generated for use by Sweeper.

0002DF**00**-0000-0000-C000-000000000046 through 0002DF**FF**-0000-0000-C000-000000000046

Any allocations out of this range should be documented in this appendix.

GUID	Symbolic Name	Description
0002DF**00**-...	n/a	Type Library ID for the Sweeper Type Library
0002DF**01**-...	CLSID_IExplorer	The class ID for IExplorer
0002DF**02**-...	CLSID_StdHlinkHistory	The class ID for the History object.
0002DF**03**-...	CLSID_StdHTMLDocObject	The class ID for the HTML DocObject object.
0002DF**04**-...	IID_DInternetHistory	Implemented by CLSID_StdHlinkHistory
0002DF**05**-...	IID_DExplorer	The primary dispinterface for the Internet Explorer Frame.
0002DF**06**-...	IID_DExplorerEvents	The primary source dispinterface for the Internet Explorer Frame (it's events).
0002DF**07**-...	IID_DHyperLink	Automation compatible interface to a hyperlink object.
0002DF**08**-...	typedef enum HlinkConstants	The HLID constants.
0002DF**09-**...	CLSID_StdHyperLink	The class ID for the standard HyperLink object.
0002DF**0A-**...	IID_DInternetSettings	Automation compatible interface to the system wide Internet settings.
0002DF**0B-**...	IID_DInternetMediaTypes	Automation compatible interface to the system wide Internet media types settings (MIME types).
0002DF**0C-**...	CLSID_StdInternetMediaTypes	The class ID for the standard media types management object.
0002DF**0D-**...	IID_DInternetMediaType	Automation compatible interface for a media type object.