Creating Browser Help Objects

by Eyal Hirsch

In this article we'll see how to write a COM object called a BHO. BHO stands for Browser Helper Object and is a Microsoft extension to its Internet Explorer browser.

A BHO is a simple COM object, which implements several predefined interfaces in order to work with the browser. Once you install your BHO, your object will receive notification of interesting events in the browser. These events include indication of when the user navigates to a new URL, when that navigation completes, when the entire HTML page is displayed in the browser, and others. You can listen to these events and act accordingly. You can tell the browser to move forward or back, to refresh, or go to the home page defined for that user.

You can use such a BHO to prevent users from accessing certain pages, and to help them reach other pages. Another possibility would be to collect information on the user's habits. Later on you'll see that you can even fill information for the user in the controls inside the HTML pages shown in the browser.

Skeleton BHO

Now let's see how this technology is implemented in the greatest tool, Delphi! First of all, since the BHO is an Automation object, we need to create one. Open Delphi and create a new ActiveX library project and save it as pSimpleBHO.dpr. Next, add a new Automation object from the same ActiveX wizard dialog. Name that automation object `InfoBHO`. Leave the default options as they are and don't add anything to the `IInfoBHO` interface, via the Type Library editor. We won't need any custom methods or properties for that object.

As with all COM objects, we need to register our object in the registry. However, since our object is not a regular COM object, but rather a BHO, we need to take an additional step in order for Internet Explorer to recognize our object. Select `Project | View Source` menu item and, at the bottom of the unit, add the lines of code shown in Listing 1. What this code does is register our object as a BHO under the correct entry in the registry. When Internet Explorer loads it checks under the following entry in the registry for registered BHOs:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
  Microsoft\Windows\
  CurrentVersion\
  Explorer\
  Browser Helper Objects\
  {xxxxxxxx-xxxx-xxxx-xxxx-
   xxxxxxxxxxxx}
```

The `{xxxx...xxxx}` notation stands for your automation object CoClass GUID. If you install my BHO you'll see the following entry in your registry:

```
HKEY_LOCAL_MACHINE\'SOFTWARE\
  Microsoft\Windows\
  CurrentVersion\
  Explorer\
  Browser Helper Objects\
  {5476D9CC-444E-11D4-ACEF-
   080000178968}
```

As far as I know there is a bug in the Internet Explorer 5 (IE5) browser with regard to BHOs. It seems that IE5 is able to load only the *first* registered BHO in the system (the BHOs are sorted in the registry by their GUIDs). Therefore, if you already have a BHO registered in your system with a lower GUID, your BHO won't be called. This bug was solved in IE5.5 and can be solved in IE5 by specifying a smaller GUID for your BHO; thus promising your BHO will be the first registered one and the one that will be used. Of course this is only a hack and should be avoided.

Implemented Interfaces

Currently we have an ActiveX library and an automation object. We've also registered our BHO in the registry. However there is still some work to be done before we can see results.

```
// In the project source file, just under the {$R *.RES}
// directive add the following lines.
var
  Handle: HKey;
  Status, Disposition: Integer;
begin
  Status := RegCreateKeyEx( HKEY_LOCAL_MACHINE, 'SOFTWARE\Microsoft\Windows\'+
    'CurrentVersion\Explorer\Browser Helper Objects\'+
    '{5476D9CC-444E-11D4-ACEF-080000178968}', 0, '', REG_OPTION_NON_VOLATILE,
    KEY_READ or KEY_WRITE, nil, Handle, @Disposition );
  if Status = 0 then begin
    Status := RegSetValueEx(Handle, PChar(''), 0, REG_SZ,
      PChar('InfoBHO'), Length('InfoBHO') + 1);
    RegCloseKey(Handle);
  end;
end.
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
type
  TInfoBHO = class(TAutoObject, IInfoBHO, IObjectWithSite, IDispatch)
  private
    BHOManager : TfrmBHOManager;
    function AdviseEvents : boolean;
    function GetParameterTypeAsString( varType : TVarType ) : String;
  protected
    WebBrowser : IWebBrowser2;
  public
    procedure Initialize; override;
    // Explorer releated methods.
    function SetSite(const pUnkSite: IUnknown ):HResult; stdcall;
    function GetSite(const riid: TIID; out site: IUnknown):HResult; stdcall;
    function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer;
      Flags: Word; var Params; VarResult, ExcepInfo, ArgErr: Pointer): HResult;
      stdcall;
  end;
```

```
// Advise events to listen for browser events.
function TInfoBHO.AdviseEvents : boolean;
var
  ConnectionPoints : IConnectionPointContainer;
  WebEvents : IConnectionPoint;
  Hr : HResult;
  dwCookie : Integer;
begin
  ConnectionPoints :=
    WebBrowser as IConnectionPointContainer;
  Hr := ConnectionPoints.FindConnectionPoint(
    DIID_DWebBrowserEvents2, WebEvents );
  WebEvents.Advise( self as IDispatch, dwCookie );
  Result := true;
end;
```

```
// SetSite method
function TInfoBHO.SetSite(const pUnkSite: IUnknown):HResult;
begin
  if ( pUnkSite <> Nil ) then begin
     WebBrowser := pUnkSite as IWebBrowser2;
     AdviseEvents;
  end;
  Result := S_OK;
end;
function TInfoBHO.GetSite(const riid: TIID; out site:
  IUnknown):HResult;
begin
  Result := S_OK;
end;
```

➤ *Listing 3*

The BHO must implement several interfaces; therefore, go to the unit where your automation object is declared and change the `TInfoBHO` object to look like Listing 2. As you can see from this, the BHO implements the following interfaces: `IInfoBHO`, `IObjectWithSite` and `IDispatch`.

The `IInfoBHO` is the one automatically generated for us. However, we won't need it and it will remain empty. The second interface the BHO supports is the `IObjectWithSite`. When IE5 loads our BHO, which it finds in the registry, it queries the object for the `IObjectWithSite` interface and calls the `SetSite` method of this interface. The browser passes this method an `IUnknown` interface, which is actually an `IWebBrowser2` interface (declared in the `SHDocVW` unit, which you must include in your `Uses` clause). We'll save this interface and I'll show you in a moment what we have to do with it. The last interface our automation object supports is `IDispatch`. Actually, the object already implements the `IDispatch` interface, but we need to override it's `Invoke` method and therefore you have to redefine it.

Listing 3 shows the three methods relevant to the `IObjectWithSite` interface. As you can see, in the `SetSite` method we save the `IWebBrowser2` interface in the `WebBrowser` data member of the class. Now we call the private `AdviseEvents` method, which is used to register our BHO object as a listener for the browser's events. I won't delve here as to what exactly the COM code does here, but basically we find the correct `IConnectionPoint`, which is `DIID_DWebBrowserEvents2` in our case, and then call the `Advise` method with ourselves as the first parameter to the method.

```
DWebBrowserEvents2 = dispinterface
  ['{34A715A0-6587-11D0-924A-0020AFC7AC4D}']
  procedure StatusTextChange(const Text: WideString); dispid 102;
  procedure ProgressChange(Progress: Integer; ProgressMax: Integer); dispid 108;
  procedure CommandStateChange(Command: Integer; Enable: WordBool); dispid 105;
  procedure DownloadBegin; dispid 106;
  procedure DownloadComplete; dispid 104;
  procedure TitleChange(const Text: WideString); dispid 113;
  procedure PropertyChange(const szProperty: WideString); dispid 112;
  procedure BeforeNavigate2(const pDisp: IDispatch; var URL: OleVariant;
    var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData:
    OleVariant; var Headers: OleVariant; var Cancel: WordBool); dispid 250;
  procedure NewWindow2(var ppDisp: IDispatch; var Cancel: WordBool); dispid 251;
  procedure NavigateComplete2(const pDisp: IDispatch; var URL: OleVariant);
    dispid 252;
  procedure DocumentComplete(const pDisp: IDispatch; var URL: OleVariant);
    dispid 259;
  procedure OnQuit; dispid 253;
  procedure OnVisible(Visible: WordBool); dispid 254;
  procedure OnToolBar(ToolBar: WordBool); dispid 255;
  procedure OnMenuBar(MenuBar: WordBool); dispid 256;
  procedure OnStatusBar(StatusBar: WordBool); dispid 257;
  procedure OnFullScreen(FullScreen: WordBool); dispid 258;
  procedure OnTheaterMode(TheaterMode: WordBool); dispid 260;
end;
```

➤ *Listing 4*

After advising, our object will receive the notifications thrown by the browser. Listing 4, taken from the `SHDocVw` unit, shows the events we can listen for. The events we'll focus on here are `BeforeNavigate2` and `NavigateComplete2`. The `BeforeNavigate2` event will be fired prior to navigation in the browser, the `NavigateComplete2` will be fired once the navigation has been completed.

We override the `Invoke` method in order to catch the events the browser fires at us. The `DispID` parameter passed to the `Invoke` method tells us which event occurred. Basically, you could implement the simplest BHO to just show message boxes when events occur. You would use the following code in order to achieve this goal:

```
case DispID of
  250: ShowMessage(
    'BeforeNavigate2');
  252: ShowMessage(
    'NavigateComplete2');
end;
```

Registering The BHO

Well, your BHO should now be ready to operate. There are two approaches to registering a BHO in the system. The easiest way is to choose `Run | Register ActiveX server`. The second option is to run the regsvr32 utility with the name of your dll. Should you encounter any problems with your BHO, you can simply remove it by either choosing `Run | Unregister ActiveX server` from Delphi, or use the regsvr32 utility again, this time with the `/u` switch. Please note that I've checked my BHO only with IE5 on both NT4 and Windows 95. As far as I am aware, IE4 and prior browsers either didn't support BHOs or are buggy.

BHO Goes GUI

So far, our example BHO has been quite simple, but not too useful. However, you can very easily add some GUI to the object and therefore enable your user to have full control of the browser, and even the HTML pages inside it.

Consider a form filler utility, where you want to let your user

specify personal information to be added automatically for him or her in any relevant HTML forms. With the BHO you can identify relevant pages (based on the URL navigated to, or by examining the HTML page, for example) and pop up your GUI so that the user can click a confirmation button and you can automatically fill in the information for him or her in the browser. The form filler is just one example, you could save the entire history of the URLs this user has visited, allow filtering of which URLs can be viewed for a specific user, collect information about the content of the pages the user views... the possibilities are endless!

OK, enough said. Add a new blank form to the project and name it `frmBHOManager`. Now override the `Initialize` method and create an instance of that form and show it. Now you have a GUI opened for you by the browser.

Listing 5 shows the final code for the `Invoke` method. As you can see from the lisitng, every event has different parameters passed to it in an array of variants in the `Params` parameter. The `BeforeNavigate2` event, for example, has access to the `Post` data in the second (index 1) item of that array, the URL to navigate to is located in the sixth (index 5) item. The current `IWebBrowser2` interface, which fired the event, is passed in the seventh (index 6) item of that array. We then call a custom `public` method of the `frmBHOManager` class in order to save the `IWebBrowser2` interface for later use. I also sent a small message to a `TMemo` object in that form, which I use for debug information. The `NavigateComplete2` event has less info than the `BeforeNavigate2` event and only has the new URL and the `IWebBrowser2` interface, which fired the `NavigateComplete2` event.

That concludes the `uInfoBHO` code, which is the actual COM object acting as a BHO. Following is a discussion on the code used in the `uBHOManager` unit.

HTML Manipulated

I've created two very simple HTML pages in order to demonstrate how the BHO can manipulate an HTML page in the browser. The first HTML page is a simple form for user information. This page has three edit boxes for the user's first name, last name and email address. The fourth element is a drop down list, used to indicate the user's country. The page has a `Submit` button at the bottom of it, in order to submit the user's details. The second HTML page is shown when the user presses the `Submit` button in the first page, and simply shows a confirmation message.

In order to use this demo, you'll have to use some sort of a web server, such as PWS or IIS from Microsoft. Place the two HTML files in the root directory for your web server and, once the BHO is ready, navigate to the first HTML file (called UserInfo.htm, by the way).

The `BHOManager` has three edit boxes and one combobox, corresponding to the controls in the HTML page. Once you click the `Update HTML` button, the information you've filled in these controls will be copied into the appropriate controls in the HTML page. Listing 6 shows the two methods responsible for moving the information from your GUI into the HTML page.

The first method that is called once you click the `Update HTML` button is `btnSetValuesClick`. In this method, we call the `Get_Document` method of the `CurrentBrowser` data member, which represents the current `IWebBrowser2` object. The `Get_Document` method returns an `IDispatch` interface, which we

➤ *Listing 5*

```
// The Invoke method - where all the fun takes place.
function TInfoBHO.Invoke(DispID: Integer; const IID: TGUID;
  LocaleID: Integer; Flags: Word; var Params; VarResult,
  ExcepInfo, ArgErr: Pointer): HResult;
var
  DispParams : TDispParams;
  VarArg : TVariantArg;
  varType : TVarType;
  WebAsVariant : PVariant;
  CurrentWebBrowser : IWebBrowser2;
  PostData, TargetFrame, NavigateToUrl : String;
begin
  try
     DispParams := TDispParams(Params);
     case DispID of
       // [0]: Cancel flag  - VT_BYREF|VT_BOOL
       // [1]: HTTP headers - VT_BYREF|VT_VARIANT
       // [2]: Address of HTTP POST data  -
       //      VT_BYREF|VT_VARIANT
       // [3]: Target frame name - VT_BYREF|VT_VARIANT
       // [4]: Option flags - VT_BYREF|VT_VARIANT
       // [5]: URL to navigate to - VT_BYREF|VT_VARIANT
       // [6]: An object that evaluates to the top-level or
       //      frame
       //      WebBrowser object corresponding to the event.
       DISPID_BEFORENAVIGATE2 :
         begin
           // Get current IWebBrowser2 interface.
           VarArg := DispParams.rgvarg^[6];
           CurrentWebBrowser :=
             IDispatch(VarArg.dispVal) as IWebBrowser2;
           varType := VarArg.vt;
           GetParameterTypeAsString( varType );
           // Get PostData information.
           VarArg := DispParams.rgvarg^[2];
           PostData := VarArg.pvarVal^;
           // Get Target frame name.
           VarArg := DispParams.rgvarg^[3];
           TargetFrame := VarArg.pvarVal^;
           // Get NavigateTo url.
           VarArg := DispParams.rgvarg^[5];
           NavigateToUrl := VarArg.pvarVal^;
           if ( Assigned( BHOManager ) ) then begin
             BHOManager.SetWebBrowser( CurrentWebBrowser );
               if ( CurrentWebBrowser = WebBrowser ) then
                 BHOManager.AddLogLine(
                   'BeforeNavigate2 - TopWindow')
               else
                 BHOManager.AddLogLine(
                   'BeforeNavigate2 - FrameWindow' );
           end;
         end;
       // [0]: URL navigated to - VT_BYREF|VT_VARIANT
       // [1]: Object that evaluates to top-level or frame
       //      WebBrowser object corresponding to the event.
       DISPID_NAVIGATECOMPLETE2 :
         begin
           // Get current IWebBrowser2 interface.
           VarArg := DispParams.rgvarg^[1];
           CurrentWebBrowser := IDispatch(VarArg.dispVal) as
             IWebBrowser2;
           // Get URL navigated to.
           VarArg := DispParams.rgvarg^[0];
           NavigateToUrl := VarArg.pvarVal^;
           if ( Assigned( BHOManager ) ) then begin
             if ( CurrentWebBrowser = WebBrowser ) then
               BHOManager.AddLogLine(
                 'NavigateComplete2 - TopWindow' )
             else
               BHOManager.AddLogLine(
                 'NavigateComplete2 - FrameWindow' );
           end;
         end;
       end;
   finally
     Result := S_OK;
   end;
end;
```

```
// Update the information from the Delphi form into the HTML
// page in the browser.
procedure TfrmBHOManager.UpdateHTMLElements( UserForm :
  IHTMLFormElement );
var
  ItemIndex, ItemName : OleVariant;
  FirstNameElement, LastNameElement, EMailElement :
    IHTMLInputElement;
  CountryElement : IHTMLSelectElement;
begin
  ItemIndex := 0;
  // Set the FirstName element in the HTML page.
  ItemName := 'edtFirstName';
  FirstNameElement := UserForm.item( ItemName, ItemIndex ) as
    IHTMLInputElement;
  FirstNameElement.Set_value( edtFirstName.Text );
  // Set the LastName element in the HTML page.
  ItemName := 'edtLastName';
  LastNameElement := UserForm.item( ItemName, ItemIndex ) as
    IHTMLInputElement;
  LastNameElement.Set_value( edtLastName.Text );
  // Set the EMail element in the HTML page.
  ItemName := 'edtEmail';
  EMailElement := UserForm.item( ItemName, ItemIndex ) as
    IHTMLInputElement;
  EMailElement.Set_value( edtEMail.Text );
    // Set the Country element in the HTML page.
    ItemName := 'cbCountry';
    CountryElement := UserForm.item( ItemName, ItemIndex ) as
      IHTMLSelectElement;
    CountryElement.Set_value(cbCountry.Items[
      cbCountry.ItemIndex]);
end;
```

```
//  The SetValues button was clicked - set the values
//  specified in the BHOManager into the appropriate fields
//  in the HTML page.
procedure TfrmBHOManager.btnSetValuesClick(Sender:
  TObject);
var
  HtmlDocument : IHTMLDocument2;
  HtmlForms, HtmlFormElements : IHTMLElementCollection;
  UserForm : IHTMLFormElement;
  FormName : WideString;
  Name, Index : OleVariant;
begin
  // Check whether CurrentBrowser data member is assigned.
  if ( not Assigned( CurrentBrowser ) ) then
    exit;
  // Get HTML document held by CurrentBrowser data member.
  HtmlDocument :=
    CurrentBrowser.Get_Document as IHTMLDocument2;
  // Get collection of HTML forms in the current page.
  HtmlForms := HtmlDocument.forms;
  memDebug.Lines.Add('Form count is ' +
    IntToStr(HtmlForms.Get_length));
  Index := 0;
  Name := 0;
  // Retrieve Form indexed zero, which is only one for now
  UserForm := HtmlForms.item( Name, Index ) as
    IHTMLFormElement;
  // Get the name of the form.
  FormName := UserForm.Get_name;
  if ( FormName <> '' ) then
    memDebug.Lines.Add( 'Form name is ' + FormName );
  UpdateHTMLElements( UserForm );
end;
```

➤ *Listing 6*

typecast to an `IHTMLDocument2` interface. Next we use the `Forms` property of the `IHTMLDocument2` interface to retrieve a collection of the HTML forms in the HTML page. Since I've defined only one form in the HTML page, we can retrieve the first form element in that collection and call the `UpdateHTMLElements` method with this `IHTMLFormElement` interface as its sole parameter.

The `UpdateHTMLElements` method receives the `IHTMLFormElement` interface from which it can retrieve the different controls on the form. We use the `Item` method of the `IHTMLFormElement` interface in order to retrieve the desired element. Listing 7 shows how we retrieve the first name element control from the HTML page and set its value with the value specified by the user.

The `ItemName` is an `OleVariant` variable and we assign the `edtFirstName` value name to it, as this is the name of the first name control in the HTML page. Now the `Item` method is used to retrieve the desired object. The return value for the `Item` method is an `IDispatch` interface and therefore we need to typecast it to an `IHTMLInputElement` interface. Once we have an `IHTMLInputElement` interface we can call the `Set_Value` method on that interface with the value specified by the user.

```
// Set the FirstName element in the HTML page.
ItemName := 'edtFirstName';
FirstNameElement := UserForm.item( ItemName, ItemIndex ) as IHTMLInputElement;
FirstNameElement.Set_value( edtFirstName.Text );
```

➤ *Listing 7*

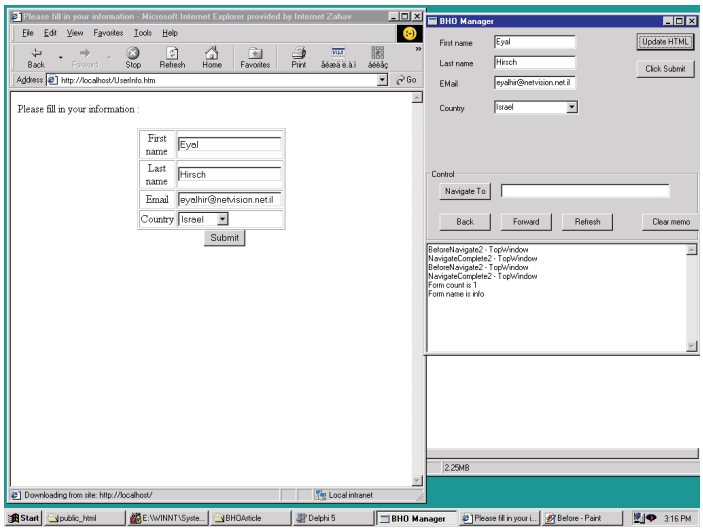
The `BHOManager` also lets you go back, forward, home and navigate to a specific URL in the browser. Please refer to the code on the disk for the entire implementation of the `BHOManager`.

One last thing to mention is that to use the HTML interface, you must add the `MSHTML` unit to your `uses` clause. I recommend you take the time to look inside this unit so that you can see the other HTML related interfaces for yourself.

Conclusion

Figure 1 shows the BHO after I've clicked the `Update HTML` button with my information.

Once you build a BHO object you are only limited by your imagination as to what you can achieve with this power. Through the BHO you can manage and listen to everything the user does in the browser, hence you have valuable information in your hands.



➤ *Figure 1*

Eyal Hirsch works as a software developer in Israel and, although he has to use Visual C++ from time to time in his work, Delphi is his avowed first love! Email Eyal at eyalhir@netvision.net.il