How to call Delphi code from scripts running in a TWebBrowser (part 5 of 6)

Case study with source code

Case study

Overview

Our case study is a simple application that illustrates the techniques discussed in this article.

The program we will develop lists some of the programs available on the *DelphiDabbler* website. Clicking one of the program names will display a brief description (precis) of the program in a box that follows list of programs. When the mouse is moved over a program name the URL of the program's web page will be displayed in the status bar. The status bar will clear when there is no program name under the mouse. The precis of each program is stored in a data file that is read by our application.

Here is a screenshot of the completed program:

Figure 1: Screenshot of case study program

We will develop the program in the following order:

1. Design the main form.
2. Define the required external object and create its type library / interface.
3. Implement the external object.
4. Implement the *IDocHostUIHandler* interface's *GetExternal* method.
5. Register the *IDocHostUIHandler* implementation with the web browser control.
6. Create the HTML file, containing the required JavaScript, that will be displayed in the browser control.

Designing the main form

To begin the project, start a new Delphi GUI application and set up the main form as follows:

- Drop a *TStatusBar* and set its *SimplePanel* and *AutoHint* properties to *True*.
- Drop a *TWebBrowser* control and set its *Align* property to *alClient*.

That is all there is to the main form.

Defining the external object

Let us consider the methods we need to add to the browser's *external* object. From the specification above we see that we need methods to perform the following actions:

1. Get the precis of a specified program when a program name is clicked (*GetPrecis* method). We will use an id string to uniquely identify each program.
2. Display the URL of a program's web page in the status bar when the mouse cursor is over a program name (*ShowURL* method). Again we will pass the program's id as a parameter.
3. Clear the status bar when no program's name is under the cursor (*HideURL* method).

We can now create an interface that contains each of the required methods. Start Delphi's Type Library Editor and use it to create a new interface named *IMyExternal*. Now add three methods using the information in *Table 2*.

IMyExternal Methods

Name	Params	Types	Modifier
GetPrecis	ProgID Result	BSTR BSTR *	[in] [out,retval]
ShowURL	ProgID	BSTR	[in]
HideURL	-	-	-

Table 2

Press CTRL+S to save the type library and name it `Article22.tlb`. Delphi will now create a type library file of that name and a Pascal unit named `Article22_TLB.pas`. Opening `Article22_TLB.pas` will reveal the *IMyExternal* interface declaration shown in *Listing 10*. (Note that the GUID will be different).

```
type
  ...
  IMyExternal = interface(IDispatch)
    ['{4F995D09-CF9E-4042-993E-C71A8AED661E}']
    function GetPrecis(const ProgID: WideString): WideString;
      safecall;
    procedure ShowURL(const ProgID: WideString); safecall;
    procedure HideURL; safecall;
  end;
  ...
```

Listing 10

Implementing the external object

Now that we have created the *IMyExternal* interface, we implement it in a class named *TMyExternal*. *Listing 11* has the class declaration.

```
type
  TMyExternal = class(TAutoIntfObject, IMyExternal, IDispatch)
  private
    fData: TStringList; // info from data file
    procedure ShowSBMsg(const Msg: string); // helper method
  protected
    { IMyExternal methods }
    function GetPrecis(const ProgID: WideString): WideString;
      safecall;
    procedure ShowURL(const ProgID: WideString); safecall;
    procedure HideURL; safecall;
  public
    constructor Create;
    destructor Destroy; override;
  end;
```

Listing 11

As expected, the methods of *IMyExternal* are specified in the class's protected section. We also have a private helper method – *ShowSBMsg* – that displays a given message in the status bar. This method is used by both *ShowURL* and *HideURL* as we will see in a moment. The *fData* string list is used to store the precis of the different programs. This field is accessed by *GetPrecis*.

Let us look at the implementation of the class. We will start with the constructor and destructor shown in *Listing 12*.

```
constructor TMyExternal.Create;
var
  TypeLib: ITypeLib;    // type library information
  ExeName: WideString;  // name of our program's exe file
begin
  // Get name of application
  ExeName := ParamStr(0);
  // Load type library from application's resources
  OleCheck(LoadTypeLib(PWideChar(ExeName), TypeLib));
  // Call inherited constructor
  inherited Create(TypeLib, IMyExternal);
  // Create and load string list from file
  fData := TStringList.Create;
  fData.LoadFromFile(ChangeFileExt(ExeName, '.dat'));
end;

destructor TMyExternal.Destroy;
begin
  fData.Free;
  inherited;
end;
```

Listing 12

The first three executable lines in the constructor are boilerplate code that has been explained earlier in the article. Following the call to the inherited constructor we create the *TStringList* object that is to store the precis data. We then read the string list's contents from a data file named `Article22.dat` that is expected to be found in the same directory as the application. The format of the data file is described later. The destructor simply frees the string list.

Now move on to examine the implementation of the three *IMyExternal* methods shown in *Listing 13*.

```
function TMyExternal.GetPrecis(
  const ProgID: WideString): WideString;
begin
  Result := fData.Values[ProgId];
end;

procedure TMyExternal.HideURL;
begin
  ShowSBMsg('');
end;

procedure TMyExternal.ShowURL(const ProgID: WideString);
begin
  ShowSBMsg(
    'http://www.delphidabbler.com/software?id=' + ProgID
  );
end;
```

Listing 13

GetPrecis looks up the given *ProgID* in the *Values* property of *fData* and returns the value found there. The data file that was loaded into *fData* in the constructor contains a line for each program. Each line has the format `ProgID=Precis`. The *TStringList.Values[]* property is designed to work with strings in this format.

HideURL uses *ShowSBMsg* to display an empty string in the status bar, which has the effect of clearing any previous message.

ShowURL simply appends its *ProgID* parameter to a literal string to produce the URL of the program's web page. It then calls *ShowSBMsg* to display the URL in the status bar.

All that remains is to look at *Listing 14*, which shows the implementation of *ShowSBMsg*.

```
procedure TMyExternal.ShowSBMsg(const Msg: string);
var
  HintAct: THintAction;
begin
  HintAct := THintAction.Create(nil);
  try
    HintAct.Hint := Msg;
    HintAct.Execute;
  finally
    HintAct.Free;
  end;
end;
```

Listing 14

Hmm – no mention of the status bar! What's happening here is that we're creating an instance of the VCL's *THintAction* action class, storing the message we want to display in its *Hint* property then executing the action. A magical feature of *THintAction* is that it automatically displays its hint in any *TStatusBar* that has its *AutoHint* property set to *True*. This let's us decouple our *external* object implementation quite nicely from the program's form.

Implementing IDocHostUIHandler

As already noted, we are re-using code from an earlier article for our declaration of *IDocHostUIHandler* and for the do-nothing implementation of the interface, *TNulWBContainer*. So, to begin with, we must add the `IntfDocHostUIHandler.pas` and `UNulContainer.pas` units, developed in that article, to our project.

Unit source code

Units containing *IDocHostUIHandler* & *TNulWBContainer* are included with this article's source code.

We now create our custom container class, *TExternalContainer*, by descending from *TNulWBContainer* and overriding the *GetExternal* method to get the functionality we need. We will use exactly the same code as we developed in *Listings 4 & 5*.

Registering the external object

Our final piece of Delphi code registers our *TExternalContainer* object as a client site (container) for the browser control. This is done in the main form simply by instantiating a *TExternalContainer* object and passing a reference to the browser control to its constructor. Recall that *TExternalContainer*'s inherited constructor automatically registers the object as a client site of the contained web browser control.

We will use the form's *OnShow* event handler to create *TExternalContainer*. We will also use this event handler to load the required HTML file, as *Listing 15* shows.

```
procedure TForm1.FormShow(Sender: TObject);
begin
  fContainer := TExternalContainer.Create(WebBrowser1);
  WebBrowser1.Navigate(
    ExtractFilePath(ParamStr(0)) + 'Article22.html'
  );
end;
```

Listing 15

Notice that we have stored a reference to the container object in a field named *fContainer*. Add such a field, with type *TExternalContainer* to the form's declaration.

Having created the container object we must also ensure it gets freed. This is done in the form's *OnHide* event handler as *Listing 16* illustrates.

```
procedure TForm1.FormHide(Sender: TObject);
begin
  fContainer.Free;
end;
```

Listing 16

Creating the HTML file

All that remains to be done is to create the HTML file that we loaded in *Listing 15*. This file is named `Article22.html` and *listing 17* shows the file in full.

```
<?xml version="1.0"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
  <head>
    <title>DelphiDabbler Articles</title>
    <style type="text/css">
      <!--
      body {font-family: Tahoma; font-size: 10pt;}
      h1 {font-size: 12pt;}
      #precis {
        border: 1px solid silver;
        padding: 4px;
      }
      -->
    </style>
    <script type="text/javascript">
      <!--
      function ShowPrecis(progID) {
        precisObj = document.getElementById("precis");
        progObj = document.getElementById(progID);
        precisObj.innerHTML = progObj.innerHTML.bold()
          + '\<br />'
          + external.GetPrecis(progID);
      }
      //-->
    </script>
  </head>
  <body>
    <h1>DelphiDabbler Program Descriptions</h1>
    <ul>
      <li><a href="javascript:void(0);" id="codesnip"
        onclick="ShowPrecis('codesnip');"
        onmouseover="external.ShowURL('codesnip');"
        onmouseout="external.HideURL()";
        >CodeSnip Database Viewer</a>
      </li>
      <li><a href="javascript:void(0);" id="htmlres"
        onclick="ShowPrecis('htmlres');"
        onmouseover="external.ShowURL('htmlres');"
        onmouseout="external.HideURL()";
        >HTML Resource Compiler</a>
      </li>
      <li><a href="javascript:void(0);" id="chi"
        onclick="ShowPrecis('chi');"
        onmouseover="external.ShowURL('chi');"
        onmouseout="external.HideURL()";
        >Component Help Installer</a>
      </li>
      <li><a href="javascript:void(0);" id="vis"
        onclick="ShowPrecis('vis');"
        onmouseover="external.ShowURL('vis');"
        onmouseout="external.HideURL()";
        >Version Information Spy</a>
      </li>
    </ul>
    <div id="precis">
      Click a program name to see its description here.
    </div>
  </body>
</html>
```

Listing 17

Do-nothing links

Looking at the body section of the file we see that it contains a list of four program names, each defined as links (`<a>` tags) that reference no URL. Every link has a unique *id* attribute that identifies the program to which it refers.

To ensure that a link does nothing when clicked we call *javascript:void(0)* in the a-link tag's *href* attribute.

The a-links' *onclick* events call the *ShowPrecis* JavaScript routine passing in the id of the relevant program as a parameter. *ShowPrecis* is defined in the HTML head section. The function first finds the `<div>` tag with the id of "precis" and then finds the a-link element associated with the program id. The HTML enclosed by the "precis" `<div>` tag is then replaced by HTML comprising of the program name in bold, a line break and the actual precis of the program. The precis is returned by the *external.GetPrecis* method, which executes *TMyExternal.GetPrecis* in the Delphi code.

Returning to the link tags, note that the *onmouseover* events directly call *external.ShowURL* with the id of the required program while the *onmouseout* events call *external.HideURL*. These JavaScript methods execute methods of the same name in *TMyExternal*, which in turn show and hide the program's URL in the status bar.

The only other item of note in the HTML file is that the head section contains an embedded style sheet that styles the `<body>`, `<h1>` and `<div id="precis">` elements.

This description of the HTML file completes the discussion of the case study.

Source code

The case study is available for download. Delphi 7 was used to create the program and it was tested on Windows XP Pro SP2 using Internet Explorer 6.

A *ReadMe* file that describes how to use the source is included in the download.

This source code is merely a proof of concept and is intended only to illustrate this article. It is not designed for use in its current form in finished applications. The code is provided on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. The source code is released under the same Creative Commons license as this article – see the bottom of this page for details. If you agree to all this then please download the code using the following link.

Download the source code

Delphi 2007 users: I've had a report that the demo code does not compile with Delphi 2007 unless you rename the unit `Article22_TLB.pas` to `Article22TLB.pas` and change references to it accordingly.

That completes the substantive part of the article. The *final section* summarises what we've achieved and provides links to some reference material.
