Implementing Active Server Pages

by Eyal Hirsch

Writing internet applications using Delphi is a snap ever since Delphi 3 was released. Thanks to the new technologies presented in Delphi 3 we can develop ISAPI, NSAPI, CGI and WinCgi applications with greater ease than ever before. We can also deploy ActiveX forms and use DCOM, Midas and CORBA in order to distribute code and objects over the net. However, one technology with no apparent support in Delphi is Microsoft's Active Server Pages (ASP).

As we'll see in this article, though, you can easily harvest the power of ASP in your Delphi apps.

ASP In General

Active Server Pages enable us to use objects installed on the web server machine, from within the client's browser. They also enable us to call methods and manipulate properties defined by these objects. Basically, we use special tags in the HTML file, so that when the web server program (be it the Personal Web Server, PWS, or the Internet Information Server, IIS) parses the HTML file and encounters these tags our object will be instantiated. Inside our object, we have the opportunity to retrieve data from the client's browser and send data back to the browser, thanks to a set of predefined interfaces supplied to us by the web server. Using Delphi's built-in support for Automation objects and ActiveX libraries, we can now build ASP objects with all the ease and flexibility that Delphi can offer.

Note that since the ActiveX library, which will host our objects, will be implemented as a DLL, the web server and the DLL will share the same processing space. Thus, the DLL will be available when objects within it are called repeatedly from within an ASP file. This is a great advantage over CGI applications, where the server has to load the CGI program each time it is called. In ASP applications the DLL is only loaded once. So ASP has a speed advantage over CGI, but this also means that debugging the DLL means restarting the web server, as for ISAPI and NSAPI but not for CGI.

Preliminary Actions

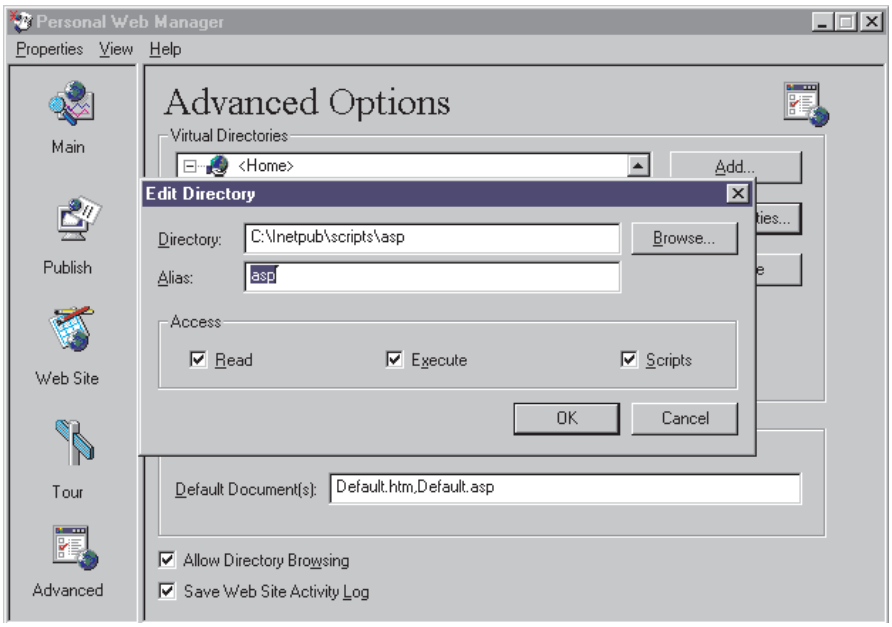
Before we can start to use ASP we have to take some preliminary actions. First, we must install a web server on our machine. We can install the PWS on a Win95/98 system or use IIS4 on Windows NT. You can download both the servers from Microsoft's web page. In this article I'll concentrate on PWS for Win95/98, as there are no significant differences between that and IIS. What I describe here should also work on IIS.

The next step is to define a virtual directory for the ASP files (I'll place the ASP and HTML files in the same directory). Start PWS and click the `Advanced` icon. In the `Advanced` form, click the `Add` button, this in turn will open a form where you specify the location of your virtual directory, its alias name and the rights given for this alias. I've created a directory named ASP under C:\InetPub\Scripts and named the alias ASP. Also check all the checkboxes in the `Access` panel. See Figure 1 for a snapshot of this screen. After clicking OK, you should be able to see the /ASP alias in the `Virtual Directories` treeview. All the ASP and HTML files should go underneath this directory.

In order to be sure everything is defined as it should be, place an empty HTML file named default.html in this directory and click the `Start` button in PWS's main page. Run Internet Explorer and in the `Address` combobox type http://127.0.0.1/asp. This should display the default empty HTML file.

When using ASP, you must specify your computer name: either the name of your computer on the net (eg http://MyComputer/asp) or a fully qualified URL such as http://www.mycompany.com/asp. The 127.0.0.1 address is a default address pointing back to your computer: throughout this article and also in my HTML files I'll use this address. As far as I'm aware, Delphi 4 currently has no support for debugging ASP files, so there are no other actions to take in order to enable debugging, as

➤ *Figure 1*

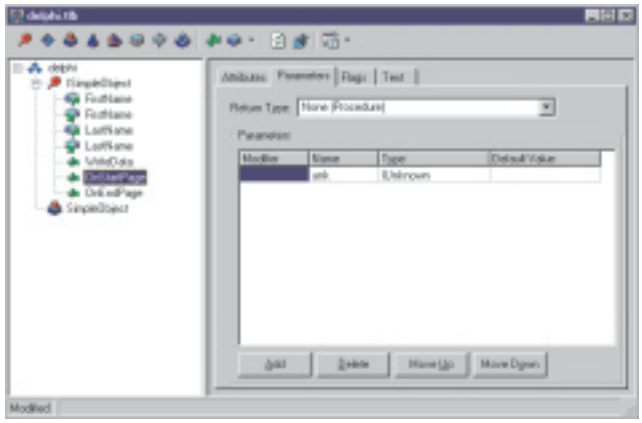


opposed to ISAPI/NSAPI and CGI applications. Once PWS runs an ASP file and our object is created, you won't be able to recompile the DLL again. The reason for that is that PWS loads this DLL into memory and unless you can unload it, you won't be able to recompile. You can unload the DLL by running the PWS program with the /stop switch from Window's Run command option. Strangely enough, pressing the Stop button in PWS won't unload the DLL, so you must stop it manually or reboot the machine (keep up the good work Microsoft!).

Getting Started With ASP

Let's open Delphi and start writing ASP objects. Close any open projects, choose the File | New menu item, go to the ActiveX page and double click the ActiveX library icon. Save the project anywhere you want with the name Delphi. Now click Project | Import type library and choose Microsoft Active Server Pages (Version 2.0) and click OK. Add the ASPTypeLibrary_TLB.pas file created for you (the default is in the Delphi4/Imports directory) into the project. This file contains all the ASP-related objects declared by Microsoft, which enable us to respond to user activities. Choose File | New again and double click the Automation object in the ActiveX page, naming the class SimpleObject.

➤ *Figure 2*



Delphi will open the type library editor form, where you can see your project name as the root of the TypeLib and the ISimpleObject interface followed by the SimpleObject co-class. Add two WideString properties to the ISimpleObject interface and name them FirstName and LastName (both are read and write properties). Next add the methods WriteData, OnStartPage and OnEndPage. Finally, add a constant parameter named unk of type IUnknown to the OnStartPage method.

Your type library should now look like Figure 2.

When PWS encounters a reference to an automation object in the ASP file (we'll see how to do this shortly) it creates the object and calls our OnStartPage function, passing it an interface reference of type IUknown, which we'll cast into an IScriptingContext interface (declared in the ASPTypeLibrary_TLB file) and save it in a private field in the automation object, named fScript. This field will enable us to interact with the browser later on in the WriteData method. The last thing to do is to override the Initialize method of the TAutoObject class, so we can initialize some of our fields. In addition, for this specific object, we'll create a custom form, which you can use for any desired task you can think of. I didn't use it in this example; however, one

➤ *Listing 1*

```
unit SimpleObject;
interface
uses
  ComObj, ActiveX, delphi_TLB, ASPTypeLibrary_TLB,
  uServerForm;
type
  TSimpleObject = class(TAutoObject, ISimpleObject)
  private
    fScript : IScriptingContext;
    fFirstName, fLastName : WideString;
  protected
    function Get_LastName: WideString; safecall;
    function Get_FirstName: WideString; safecall;
    procedure OnEndPage; safecall;
    procedure OnStartPage(const unk: IUnknown); safecall;
    procedure Set_LastName(const Value: WideString);
      safecall;
    procedure Set_FirstName(const Value: WideString);
      safecall;
    procedure WriteData; safecall;
  public
    procedure Initialize; override;
  end;
implementation
uses ComServ;
function TSimpleObject.Get_LastName: WideString;
begin
  Result := fLastName;
end;
function TSimpleObject.Get_FirstName: WideString;
begin
  Result := fFirstName;
end;
procedure TSimpleObject.OnEndPage;
begin
  fScript := nil;  // Release IScriptingContext interface
end;
procedure TSimpleObject.OnStartPage(const unk: IUnknown);
begin
  // Save the IScriptingContext interface
  fScript :=  unk as IScriptingContext;
end;
procedure TSimpleObject.Set_LastName(
  const Value: WideString);
begin
  fLastName := Value;
end;
procedure TSimpleObject.Set_FirstName(
  const Value: WideString);
begin
  fFirstName := Value;
end;
procedure TSimpleObject.WriteData;
begin
  // Write response page to client's browser.
  fScript.Response.Write('<TABLE>' );
  fScript.Response.Write('<TR><TD>First name : </TD>');
  fScript.Response.Write('<TD>'+fFirstName+'</TD></TR>');
  fScript.Response.Write('<TR><TD>Last name : </TD>');
  fScript.Response.Write('<TD>'+fLastName+'</TD></TR>');
  fScript.Response.Write('</TABLE>');
end;
procedure TSimpleObject.Initialize;
begin
  inherited Initialize;  // Initialize the object's state
  fFirstName := '';
  fLastName := '';
  // Create the server form.
  if not Assigned(frmAspServer) then begin
    frmAspServer := TfrmAspServer.Create ( nil );
    frmAspServer.Show;
  end;
end;
initialization
  TAutoObjectFactory.Create(ComServer, TSimpleObject,
    Class_SimpleObject, ciMultiInstance, tmApartment);
end.
```

implementation I can think of is to add the names of the users to a string grid control. The complete source is shown in Listing 1.

Three important methods to notice in Listing 1 are OnStartPage, OnEndPage and WriteData. Inside OnStartPage we save the IScriptingContext interface we've received from PWS into the private field named fScript. In the OnEndPage method we release this interface, by assigning it a nil value (it's important to remember to always clean up after yourself!).

The WriteData method uses the fScript field to write data back into the client's browser. We use the Response property of the IScriptingContext interface, which is of type IResponse, along with the Write method of this interface, to write the first and last name of the user back into the client's browser.

We're now only a few mouse clicks away from completing the Delphi code. All that is left now is to click the Run | Register ActiveX Server menu item, which will compile the project into a DLL, and register our type library and automation object with the Windows registry.

Creating SimpleObject HTML And ASP Files

Next we'll create the HTML file where the user can supply his or her data. This is a simple HTML file with some text and two edit boxes for the users to supply their first and last name (see Listing 2).

The most important line in the SimpleObject.HTML file is this:

```
<form Action="http://127.0.0.1/
  asp/SimpleObject.asp"
  METHOD="post">
```

We encapsulate all of our HTML controls inside this form tag. The form's method is POST and its Action field dictates what page should be processed next, in this case the SimpleObject.ASP file, which is placed in our virtual directory, ASP. When the user clicks the Submit button the ASP file will be processed by the web server, which will eventually cause our ASP object to be instantiated.

Let's create our first ASP file now. This is a simple text file, which you can create with any text editor (such as NotePad) or with a tool such as Microsoft's FrontPage. All you have to remember when writing ASP files is that ASP calls are enclosed by the <% characters; otherwise they are merely HTML files. This means that you can use ASP calls and HTML tags interchangeably.

First we have to create our Automation object from the ASP file. We accomplish this by using the Server object and calling its CreateObject method, passing it the object's name, in this case delphi.SimpleObject. Next, we assign the data typed by the user to the automation object's properties, using the Request object, which is of type IRequest. The Request object has a property named Form, which is like an array property in Delphi. We use the Form property, in order to retrieve the data supplied by the user, in the following manner:

```
objDelphi.FirstName =
  Request.Form("firstname")
```

Now all that is left to do, is call our object's WriteData method and we are done. You can see the ASP code in Listing 3. The resulting HTML is shown in Figure 3.

Databases And ASP

Being able to save user data or supply the user with data from a remote database is a major issue when implementing internet applications. Since the article is about ASP and we are using Delphi to implement the ASP objects, why not use Delphi's capabilities in the database area? Nothing prevents us from adding a data module to the type library, along with tables, queries and other native database components provided by Delphi.

The next ASP object we'll build will enable us to retrieve data from the demo tables supplied with Delphi. Run Delphi and re-open the last project (delphi.dpr). Add a new data module to the project, add to it a TQuery component, name it qryTable, set its DataBase property to DBDEMOS and clear its SQL property. Next, add another Automation object to the project and name it SQLObject.

Now we can start adding methods and properties to this object. First we create the

➤ *Above: Listing 2*

```
<html>
<head>
<title>Simple Object page</title>
</head>
<body>
<form ACTION="http://127.0.0.1/asp/SimpleObject.asp" METHOD="post">
  <p>Enter the following details</p>
  <table border="0" width="381">
    <tr>
      <td width="91">First name :</td>
      <td width="201"><input TYPE="text" SIZE="20" NAME="firstname"> </td>
    </tr>
    <tr>
      <td width="91">Last name :</td>
      <td width="201"><input TYPE="text" SIZE="20" NAME="lastname"> </td>
      <td width="77"><input TYPE="submit" VALUE="submit"> </td>
    </tr>
  </table>
</form>
</body>
</html>
```

➤ *Below: Listing 3*

```
<html>
<head>
<title>Using Simple Object</title>
</head>
<body>
<%
  'Create the object named SimpleObject.
  set objDelphi = Server.CreateObject("delphi.SimpleObject")
  'Set the properties of that object.
  objDelphi.FirstName = Request.Form("firstname")
  objDelphi.LastName =Request.Form("lastname")
  'Display the data supplied by the user.
  objDelphi.WriteData
%>
</body>
</html>
```

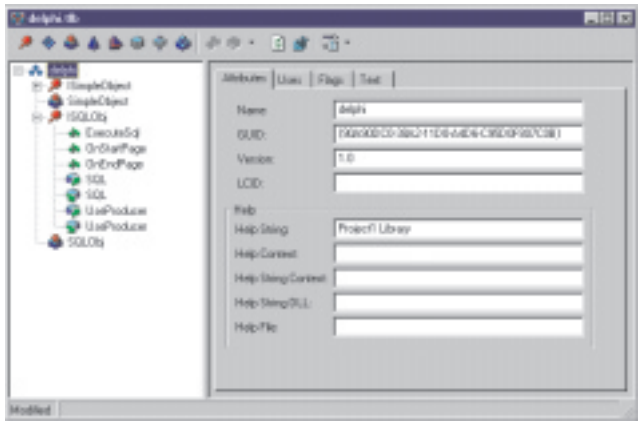
➤ *Figure 3*

methods OnStartPage and OnEndPage, just as we've done with the last object (simply copy it from the previous object).The implementation of these methods is exactly the same as in the SimpleObject, so I won't delve into explaining these methods again. Add an SQL property of type WideString and an integer property named UseProducer. Both properties are read and write properties. Finally add the ExecuteSql method. Your type library should look like Figure 4 by now.

Take a look at Listing 4 which shows the relevant code for the SqlObject. Notice that most of the code is quite straightforward and repeats itself. The OnStartPage and OnEndPage methods are the same as in the SimpleObject object and the Initialize procedure is almost identical. Note that I have also declared two additional private fields in order to save the SQL sentence that the user specified and whether he or she wants to use the TDataSetTableProducer component.

As you have probably noticed, our goal in ASP is to retrieve user input from within his or her browser and act upon it. We manipulate the user's input and then write some HTML text back to the client's browser. It is up to us to decide how this HTML text will be produced. We can build this HTML text manually, as we will do when the UseProducer checkbox is unchecked, or we can do it with components. When the user checks the UseProducer checkbox, we'll use the TDataSetTableProducer component provided by Delphi to produce the HTML output.

The action takes place in the ExecuteSql method of the SQLObject class. In this method, we add the value of the private field named fSQL into the SQL property of the TQuery component. The fSQL property is set in the Set_SQL method and holds the SQL sentence specified by the user in the browser. Next we open the query. Using the IScriptingContext interface given to us in the OnStartPage method, we write the record count property of the query, courtesy of the Write method of the Response object. Next, we'll report to the user the state of the UseProducer checkbox. As you can see, we can actually check the value of a control placed in the HTML file from within Delphi. The IScriptingContext interface enables us to



➤ *Figure 4*

➤ *Listing 4*

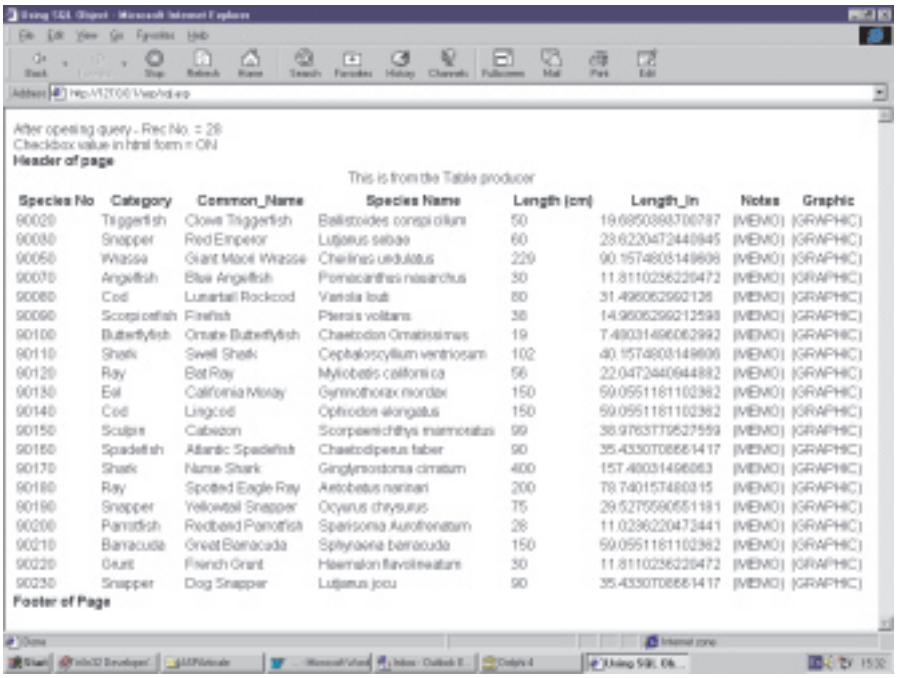
```
type
  TSQLObj = class(TAutoObject, ISQLObj)
  private
    fScript : IScriptingContext;
    fSql : WideString;
    fUseProducer : Integer;
    procedure BuildTableData;
  protected
    procedure ExecuteSql; safecall;
    procedure OnStartPage(const unk: IUnknown); safecall;
    procedure OnEndPage; safecall;
    function Get_SQL: WideString; safecall;
    procedure Set_SQL(const Value: WideString); safecall;
    function Get_UseProducer: Integer; safecall;
    procedure Set_UseProducer(Value: Integer); safecall;
  public
    procedure Initialize; override;
  end;
procedure TSQLObj.BuildTableData;
var i : integer;
begin
  // Build html response page to be sent to client's browser
  with fScript.Response, dmBioLife.qryTable do
    if ( fUseProducer = 0 ) then begin
      while not Eof do begin
        // Iterate through the data set
        // Write field's values to the browser
        for i := 0 to Pred(dmBioLife.qryTable.FieldCount) do
          Write(Format('%s, ',
            [dmBioLife.qryTable.Fields[i].AsString]));
        Next;
        Write('<br>');
      end;
    end else
      // Display the data using the page producer
      write(dmBioLife.dsProducer.Content);
end;
procedure TSQLObj.ExecuteSql;
var sValue : string;
begin
  try
    // Add the client's sql sentence.
    if ( Length ( fSql ) > 0 ) then begin
      dmBioLife.qryTable.Sql.Clear;
      dmBioLife.qryTable.Sql.Add ( fSql );
    end;
    dmBioLife.qryTable.Active := true;
    with fScript.Response, dmBioLife.qryTable do begin
      // Report back to user some custom data.
      Write(Format(
        'After opening query - Rec No. = %d<br>',
        [dmBioLife.qryTable.RecordCount]));
      sValue := UpperCase(
        fScript.Request.Form.Item['cbDsProducer']);
      if(sValue = '') then
        sValue := 'OFF';
      Write(Format('Checkbox value in html form = %s<br>',
        [sValue]));
      // Build html response page to send to browser
      BuildTableData;
    end;
  finally
  end;
end;
```

```
<html>
<head>
<title>Sql Object page</title>
</head>
<body>
<form ACTION="http://127.0.0.1/asp/SqlObject.asp" METHOD="POST">
  <table border="0" width="450">
    <tr>
      <td>SQL line</td>
      <td><input TYPE="text" VALUE="select * from biolife" SIZE="20"
        NAME="edtSql"> </td>
      <td><input TYPE="checkbox" VALUE="on" NAME="cbDsProducer">
         Use Dataset Producer </td>
    </tr>
    <tr>
      <td></td>
      <td><input TYPE="submit" VALUE="Submit" NAME="btnSubmitQuery">
        <input TYPE="reset" VALUE="Reset" NAME="B2"> </td>
    </tr>
  </table>
</form>
</body>
</html>
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
<html>
<head>
<title>Using SQL Object</title>
</head>
<body>
<%
  'Create the object named SQLObject
  set objDelphiSql = Server.CreateObject("delphi.SQLObj")
  'Set the SQL property to the value specified by the user
  objDelphiSql.SQL = Request.Form("edtSql")
  if Request.Form("cbDsProducer")="on" then
   objDelphiSql.UseProducer = 1
  else
   objDelphiSql.UseProducer = 0
  end if
  'Run the query
  objDelphiSql.ExecuteSql %>
</body>
</html>
```



➤ *Figure 5*

have access to the `Form` object through its `Request` object. The `Form` object holds an arrayed property named `Item`, which in turn holds all the control's values placed on the HTML form. When the user checks this checkbox, the value of this item will be `ON`, as specified in the SqlObject.HTML file.

These two actions, reporting the record count and the value of the checkbox, are optional. I used them when debugging the project and decided to leave them in so you can see how to use them.

All that is left to do now is to send the query results back to the browser, so the user can see them. Printing the query result is done in the `BuildTableData` method. First we have to check the value of the `fUseProducer` field, if its value is other than zero then the user specified he or she wants to use the `TDataSetTableProducer` component. If the user didn't want to use the `TDataSetTableProducer` component, we simply iterate through the resulting data set and write each record in a separate line using the HTML tag `<br>`.

ASP And HTML For SqlObject

SqlObject.Asp is the ASP file for the `SqlObject`. This file is very simple: we create our object using the `Server`'s method `CreateObject` with the appropriate object name. Then we set the object's `Sql` and `UserProducer` properties. Finally we call the `ExecuteSql` method to run the query.

The HTML file for this object (SqlObject.HTML) is also simple, containing a `Submit` button, a `Reset` button, an editbox for the SQL text and a checkbox control. The code for the HTML file is in Listing 5 and the ASP file is shown in Listing 6. The resulting HTML file, which was produced with the `UseProducer` option on, is shown in Figure 5.

Summary

In this article we've seen how to implement ASP objects using Delphi. Note that I didn't delve into the ASP syntax: it is very simple and you won't have any trouble learning it as it is very similar to Delphi.

Acknowledgement

Additional information about ASP can be found at www.15seconds.com and www.activeserverpages.com. Thanks to those who assisted me getting started with ASP.

Eyal Hirsch works as a Delphi developer in Israel. He can be contacted via email as eyalhir@netvision.net.il