Dr.Bob Examines... #38

See Also: other Dr.Bob Examines columns or Delphi articles

Delphi for .NET and ASP.NET

In this article, I want to demonstrate that we can create very useful things with it already. Especially as scripting language for ASP.NET applications, the Delphi for .NET preview command-line compiler has earned my respect, and can play a big role.

Note: for this article, you again need the .NET Framework and SDK installed, as well as the Delphi for .NET preview command-line compiler that ships as separate CD in the box with Delphi 7 Studio. Note that both this original version and the update (made available on November 22nd) work just fine with the code in this article.

Configuration

In order to use Delphi for .NET as scripting language for ASP.NET - or more specifically for ASP.NET to recognize Delphi as a scripting language - we may have to do a couple of things (a bit more than outlined in John Kaster's article on Borland Developer Network I'm afraid). On my development machine, I use Windows 2000 Professional with SP3, and the .NET Framework and SP2 applied. I use Internet Information Server (IIS) as web server, which means that there's a C:\InetPub directory that contains a (virtual) directory called Scripts. In this virtual directory, I have placed a file called web.config with the contents that can be seen below:

```
<configuration>
  <system.web>
    <compilation debug="true">
       <assemblies>
          <add assembly="DelphiProvider" />
       </assemblies>
       <compilers>
          <compiler language="Delphi" extension=".pas"
            type="Borland.Delphi.DelphiCodeProvider,DelphiProvider" />
       </compilers>
    </compilation>
  </system.web>
</configuration>
```

Any ASP.NET page (with the .aspx file extension) that we want to run from the Scripts directory will now force ASP.NET to look at this web.config file. And if you try it right now, you may get a "Server Error in '/' Application" message. This message indicates that a file or assembly called DelphiProvider was not found. To make sure that ASP.NET can find it, we must create a Bin directory in the virtual directory Scripts, and copy the DelphiProvider.DLL into this Bin directory. This works on most machines, although one of my development machines still complained. In that case, I had to create the Bin directory in the root of my web server's root directory, and place the DelphiProvider.DLL in that location (so if you still get the error message, you may want to try that as well). Finally, it helps if you restart IIS, and if all else fails you may want to reboot as well (as last resort) to get it to work.
As final step, you also need to copy the 3 files from the C:\Program Files\Delphi for .NET\aspx\framework directory to the C:\WinNT\Microsoft.NET\Framework\v1.0.3705 directory (these files are needed to compile the Delphi for .NET source code that will be

generated by the DelphiProvider.DLL).

Celsius to Fahrenheit conversion

Before I want to show you how to build ASP.NET pages, I first want to spend a few minutes discussing the Delphi side of the example that I want to build today: a Celsius to Fahrenheit temperature conversion tool. Not very hard to do, but enough to show the capabilities of ASP.NET in a moment.
Let's assume we need two edit controls called edtCelsius and edtFahrenheit, as well as two buttons - the first one to convert from Celsius to Fahrenheit and the second button for the other way around. The implementation for the temperature conversion inside the button OnClick event handlers can be seen below:

```
procedure TForm.CelsiusToFahrenheitClick(Sender: TObject);
begin
  edtFahrenheit.Text := FloatToStr(9 / 5 *
    StrToFloat(edtCelsius.Text) + 32)
end;

procedure TForm.FahrenheitToCelsiusClick(Sender: TObject);
begin
  edtCelsius.Text := FloatToStr(5 / 9 *
    (StrToFloat(edtFahrenheit.Text) - 32))
end;
```

Nothing hard about it, is there? Well, just watch as we use the same code inside our ASP.NET page!

Using Delphi code with ASP.NET

Web pages made for ASP.NET end with the .aspx extension, and contain scripting code as well as HTML code. For our example, I want to use two edit controls (also called textbox controls in ASP.NET), and two buttons, each with a different OnClick event handler. Since we must respond based on a button click, we can place all these controls in a HTML form as can be seen below:

```
<form runat="server">
 <br><b>Celsius:</b>
     <asp:textbox id="edtCelsius" runat="server"/>
 <br><b>Fahrenheit:</b>
     <asp:textbox id="edtFahrenheit" runat="server"/>
 <p>
 <asp:button text="Celsius to Fahrenheit"
     OnClick="CelsiusToFahrenheitClick" runat="server"/>
 <asp:button text="Fahrenheit to Celsius"
    OnClick="FahrenheitToCelsiusClick" runat="server"/>
</form>
```

This will produce two editboxes right under each other followed by two buttons. Note that the controls are not plain HTML controls such as "edit", but are special asp controls, with the asp: prefix (asp:textbox and asp:button). Also note that we're not done yet, since we need to implement the CelsiusToFahrenheitClick and FahrenheitToCelsiusClick event handlers. For this, the signature of the event handlers has changed a bit (the Sender is of type System.Object now, and there's a second argument E of type EventArgs, but the code inside the event handlers is still the same). See below for details:

```
<%@Import Namespace="Borland.Delphi.SysUtils"%>

<script language="Delphi" runat="server">
  procedure CelsiusToFahrenheitClick(Sender: System.Object; E: EventArgs);
  begin
    edtFahrenheit.Text := FloatToStr(9 / 5 *
      StrToFloat(edtCelsius.Text) + 32)
```

```
    end;

    procedure FahrenheittoCelsiusClick(Sender: System.Object; E: EventArgs);
    begin
      edtCelsius.Text := FloatToStr(5 / 9 *
        (StrToFloat(edtFahrenheit.Text) - 32))
    end;
  </script>
```

Note that the two event handlers can use the edtFahrenheit and edtCelsius editboxes, although they are not declared "at the Delphi side", but merely in the ASP.NET HTML Form. Nevertheless, the Delphi source code unit that will be generated by the DelphiProvider.DLL will contain variable declarations for the ASP.NET controls, so the event handlers will be able to use them just as in a regular Delphi application.
You may also want to pay attention to the first line of the above listing, which shows how to import a special unit that we need (namely the Borland.Delphi.SysUtils unit), otherwise the code wouldn't compile (because FloatToStr and StrToFloat could not be found).

Putting Everything Together

Before we put the last two listings together to produce a working ASP.NET page, I want to point you to the fact that on .NET, we can use anything that's available in .NET in our Delphi for .NET code. This may sound obvious, and so is the use of FloatToStr and StrToFloat - for Delphi developers. But a C# developer would use the Convert class from the System assembly, and call Convert.ToDouble and Convert.ToString to convert strings to doubles (and back). And since the System assembly is also available in a Delphi for .NET application, we can also call the Convert.ToDouble and Convert.ToString methods, of course.
Apart from that change, I've introduced an additional control, of type asp:label, called Message, and after each conversion I set the text of the asp:label control to indicate which conversion has just been executed (may be helpful for testing or debugging purposes). Anyway, the final and complete Celsius.aspx source code can be seen below:

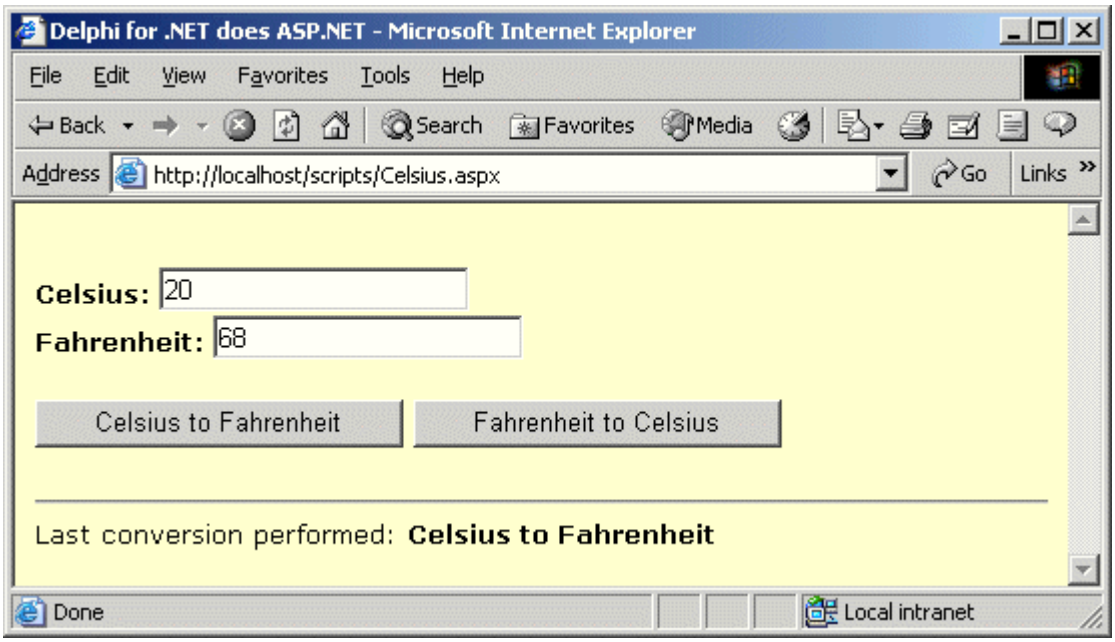
```
  <html>
  <head>
  <title>Delphi for .NET does ASP.NET</title>

  <script language="Delphi" runat="server">

    procedure CelsiusToFahrenheitClick(Sender:
      System.Object; E: EventArgs);
    begin
      edtFahrenheit.Text := Convert.ToString(9 / 5 *
        Convert.ToDouble(edtCelsius.Text) + 32);
      Message.Text := 'Celsius to Fahrenheit'
    end;

    procedure FahrenheittoCelsiusClick(Sender:
      System.Object; E: EventArgs);
    begin
      edtCelsius.Text := Convert.ToString(5 / 9 *
        (Convert.ToDouble(edtFahrenheit.Text) - 32));
      Message.Text := 'Fahrenheit to Celsius'
    end;

  </script>
  </head>

  <body bgcolor="ffffcc">
  <font face="verdana" size="2">
  <form runat="server">
   <br><b>Celsius:</b>
       <asp:textbox id="edtCelsius" runat="server"/>
```

```
  <br><b>Fahrenheit:</b>
     <asp:textbox id="edtFahrenheit" runat="server"/>
  <p>
    <asp:button text="Celsius to Fahrenheit"
     OnClick="CelsiusToFahrenheitClick" runat="server"/>
    <asp:button text="Fahrenheit to Celsius"
     OnClick="FahrenheittoCelsiusClick" runat="server"/>
</form>
<hr>
Last conversion performed:
<b><asp:label id="Message" runat="server"/></b>
</body>
</html>
```

Once we've placed this file in your Scripts directory, we can test the page by opening it with our local browser (once you install the .NET run-time or SDK, you'll be able to run ASP pages with Internet Information Server on your local machine).

Run!

Since my virtual directory is called Scripts, I open http://localhost/Scripts/Celsius.aspx, entered 20 in the Celsius editbox and clicked on the button to convert Celsius to Fahrenheit, with the results shown in Figure 1.



Note that it will take some time to execute the Celsius.aspx page for the first time only. That's because the DelphiProvider.DLL will generate Delphi source code which is then compiled to native .NET code and finally executed. After the first request, the .NET code will not be generated again (unless you've made a change to the source code), so the cached .NET code will be executed instead. This is one of the nicer features of ASP.NET, and it's very cool that we can use Delphi for .NET (even the preview!) to build ASP.NET applications today!

Temporary ASP.NET Files

If you want to examine the generated Delphi source code for the Celsius.aspx page, you either have to enable debugging by including a directive in the Celsius.aspx file, or by including a line with in the web.config file (already present in the first listing). Now you can look inside the C:\WinNT\Microsoft.NET\Framework\v1.0.3705\Temporary ASP.NET Files directory and look for subdirectories (with random names) that contain .pas files generated by the DelphiProvider.DLL.
Note that although you can read the generated files, there's no way you can edit them and

make ASP.NET use the modified source files instead. The files from the Temporary ASP.NET Files directory are just that - Temporary ASP.NET Files (but it can be insightful to take a look inside them).

More Delphi and .NET

In this short introduction to using the Delphi for .NET preview command-line compiler as scripting environment for ASP.NET pages, I hope to have given you some helpful information into what is currently possible. For more information, please make sure to check out the special Delphi for .NET website from Borland at http://dotnet.borland.com.
Finally, if you want to try the Celsius to Fahrenheit converter "live" on the Internet, feel free to surf to http://www.eBob42.com/cgi-bin/Celsius.aspx and see it in action for yourself.

This Just In...

Just when I finished the first draft of this article, the Delphi for .NET Preview command-line compiler Update #1 was released (for registered Delphi 7 users only). This version contains the core VCL for .NET functionality as well, which is something we'll check out next month. As far as I can see, VCL for .NET is meant to function as an easy way to migrate existing VCL applications from Win32 to .NET. More about this next time(s), so stay tuned...

Delphi and .NET

This ends my coverage of Delphi and the combination of COM and/in .NET. In the next few articles I will explore the capabilities of Delphi for .NET, using the preview command-line compiler to build console, visual we well as web server and web service applications, joining forces with ASP.NET when needed for the web examples. If you want to make the move with Delphi to .NET, then stay with me and be sure to get back to this website! So stay tuned, and don't hesitate to send me feedback or comments (or suggestions for topics to cover using Delphi for .NET).
