ASP.NET Web Services

by Bob Swart

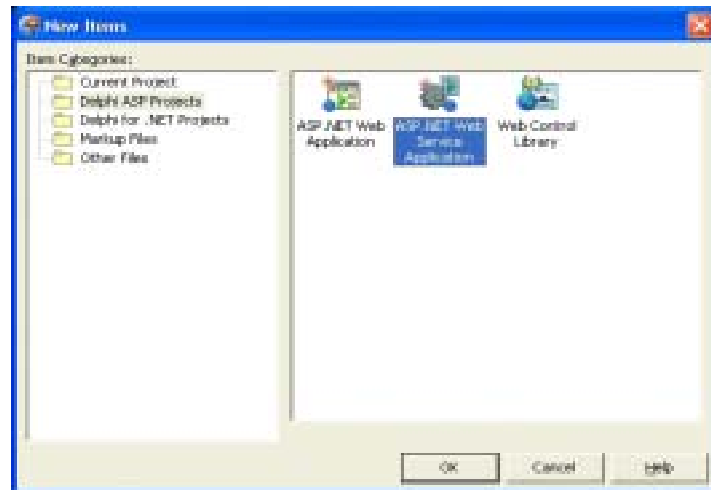
SOAP and Web Services are hot, and used just about everywhere these days. The Microsoft .NET Framework and SDK contain built-in support for web services (with the System.Web.Services.dll assembly as one of the core pieces).

This article shows in detail how we can build ASP.NET web services using Borland's Delphi for .NET, as well as how we can import web services (in this case the same one) and use them in WinForms applications - also using Delphi for .NET. It's the first time this is shown in public, so brace yourselves and prepare for a great ride!

Web Service Engine

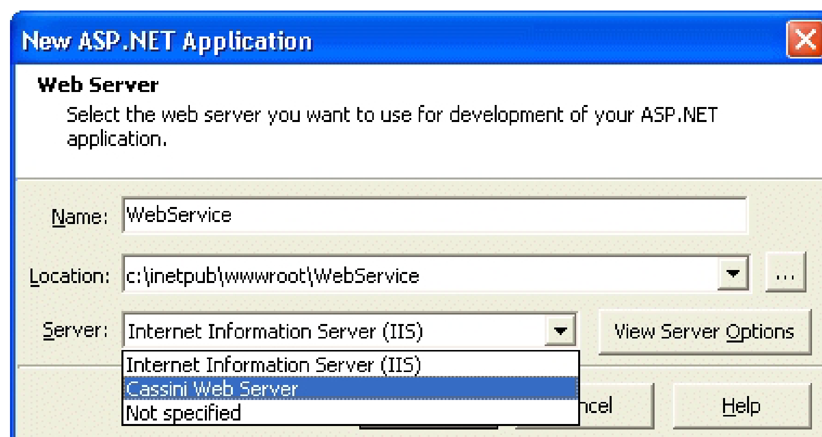
The example Web Service that we will be making and using in this article is the Centigrade to Fahrenheit Temperature Conversion Web Service, also known by the name Celsius. This Web Service is available on the web as http://www.ebob42.com/cgi-bin/Celsius42.asmx, but we'll build a fresh one using Delphi for .NET.

Start Delphi for .NET, and create a new project using File | New - Other. This will give us the Object Repository, which has three new project targets in the Delphi ASP Projects category. In time, I'll cover all possibilities of this category, but today, we'll only create an ASP.NET Web Service Application.



Double-click on the ASP.NET Web Service Application icon which starts the NEW ASP.NET Application wizard.

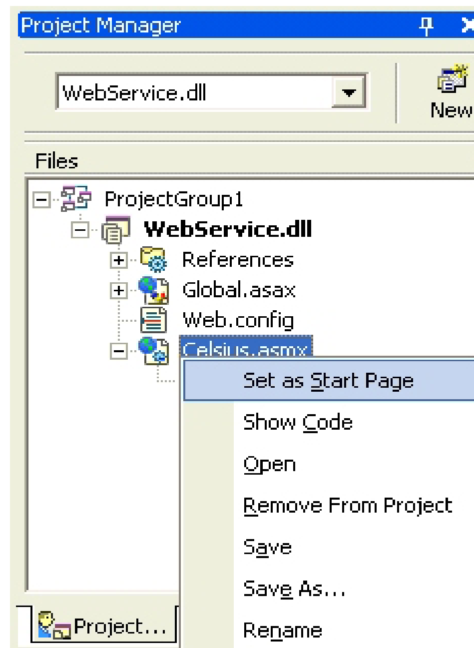
Like ASP.NET Web Forms, we can use the ASP.NET Web Service Application wizard to select a web server, with a choice between Microsoft's Internet Information Server (IIS) and the Cassini web server.



We have to specify a number of options here. First of all, the name of our project (which can contain more than one actual web service), which will also determine the namespace of the web service classes inside the project. As a second option, you can specify the location of the project. This directory - by default a subdirectory of your wwwroot directory, although you can change this setting in the ASP.NET Options page of the Tools Options dialog - will actually contain all your project files including the .asmx and .asmx.pas code-behind files. We'll take a more detailed look at these files in a moment.

Click on OK to start the new ASP.NET web service project. It consists of a project file WebService.bdsproj, WebService.cfg and WebService.dpr, and contains the files WebService1.asmx, WebService1.asmx.pas, Global.asax, Global.pas, and Web.config. The latter three file will not be covered here, since right now we only need to concern ourselves with the WebService1.asmx and .asmx.pas file, which are already opened in the code editor. Do File | Save As to save them under a new name, such as Celsius. Note that both files will be renamed at once - so I end up with Celsius.asmx and Celsius.asmx.pas.

The Delphi for .NET project options contain a reference to the .asmx page of the project to use as the start page when debugging the application. In order to specify that the renamed Celsius.asmx has to be used as the starting page, right-click on Celsius.asmx in the Project Manager and select "Set as start page" (this option is only used by the IDE, and not by the web server where you deploy the web service).



Now it's time to do File | Save All and make sure all files are saved on disk before we continue.

TCelsius Web Service

Apart from renaming the WebServices1 files to Celsius, I also want to rename the generated name of the web service class (by default TWebService1) to TCelsius. This is something we can try to do in two files: both in the new Celsius.asmx and in the Celsius.asmx.pas source file.

In the code editor, click on the tab for Celsius.asmx (which contains a single line) and replace WebService1 with TCelsius. The new contents should now be as follows:

```
<%@ WebService Language="c#" Debug="true" Codebehind="Celsius.pas"
    Class="Celsius.TCelsius" %>
```

Note that Language="c#" here, which just makes sure that the actual page that is used as generated ASP.NET web form will be a C# page, using our new assembly (in the file bin\WebService.dll) as host for the Celsius.TCelsius class.

However, if you save this file, then the name TWebService1 is put back instead of the TCelsius. This is a reminder that the .asmx file does not need to be edited by us, we should use the .asmx.pas file instead.

So, click on the tab for the Celsius.asmx.pas code behind source file. This file contains the web service class in the namespace WebService as specified in Celsius.asmx. By default that class is called TWebService1, and you have to rename it to TCelsius now (a global search and replace is the fastest way).

As soon as you save the .asmx.pas file, the .asmx file will be updated as well.

Namespace

Apart from the new name of the web service, there is one thing that you may want to add: the web service namespace information. If you don't, a web service will use the default namespace http://www.tempuri.org which is not a good idea since your web service and types should be unique - hence the need for a unique namespace. In my case, I always use http://www.eBob42.org as namespace, so just before the line which declares the web service class, I add a WebService attribute as follows:

```
[WebService(Namespace='http://www.eBob42.org',
 Description='Dr.Bob''s Celsius Web Service written in Delphi for .NET using ASP.NET')]
TCelsius = class(System.Web.Services.WebService)
```

This is enough to make sure that the TCelsius web service uses a non-default namespace. It's still an empty web service, however. If you browse through the Celsius.asmx.pas source file, you'll notice some helpful comments at the bottom of the file that contain some sample methods that you can uncomment to use. The most important thing that they should tell you is that any method that must be exported (as web service method) must be preceded by an attribute [WebMethod] in the definition of the method (i.e. inside the TCelsius class definition).

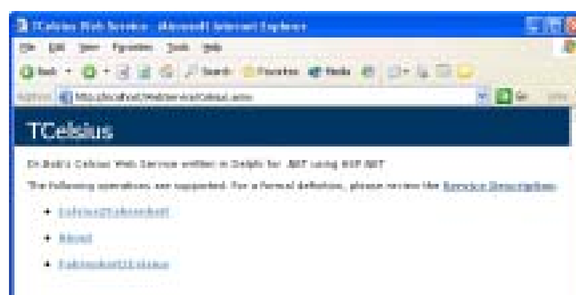
The two sample methods are HelloWorld and EchoString, and since I don't need them, just delete them and instead write three new methods: About, Celsius2Fahrenheit and Fahrenheit2Celsius. The first one has the same signature as the HelloWorld; no arguments and returning a string. The latter two have both a double argument and a double result type, converting one temperature value to another. The class definition will be as follows:

```
[WebMethod]
function About: String;
[WebMethod]
function Celsius2Fahrenheit(degrees: Double): Double;
[WebMethod]
function Fahrenheit2Celsius(degrees: Double): Double;
```

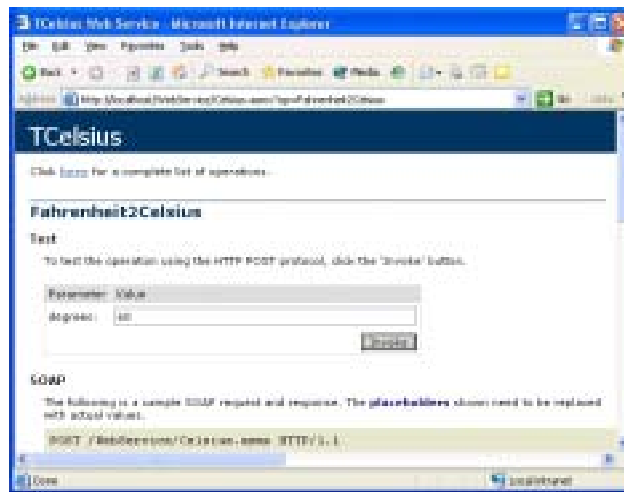
The implementatation code that you can add to the bottom of the Celsius.asmx.pas file is as follows:

```
function TCelsius.About: string;
begin
  Result := 'Dr.Bob''s Celsius Web Service written in Delphi for .NET using ASP.NET'
end;

function TCelsius.Celsius2Fahrenheit(degrees: Double): Double;
const
  AbsoluteZeroCelsius = -237.15;
begin
  if (Degrees < AbsoluteZeroCelsius) then
    raise ApplicationException.Create('Invalid Temperature');
  Result := 32 + (9 * degrees / 5)
end;

function TCelsius.Fahrenheit2Celsius(degrees: Double): Double;
const
  AbsoluteZeroFahrenheit = -394.87;
begin
  if (Degrees < AbsoluteZeroFahrenheit) then
    raise ApplicationException.Create('Invalid Temperature');
  Result := 5 * (degrees - 32) / 9
end;
```

Time to save your work and compile the project. Note that you can run it from the IDE, which means that a browser will be started and the project options will be used to determine the starting page (which we set to Celsius.asmx, remember?). To run the project without actually debugging it, do Run | Run Without Debugging. This will launch Internet Explorer and show you the information about the Celsius web service.



You can even test the methods of the web service in the browser (all facilitated by the ASP.NET environment).



With the following result:

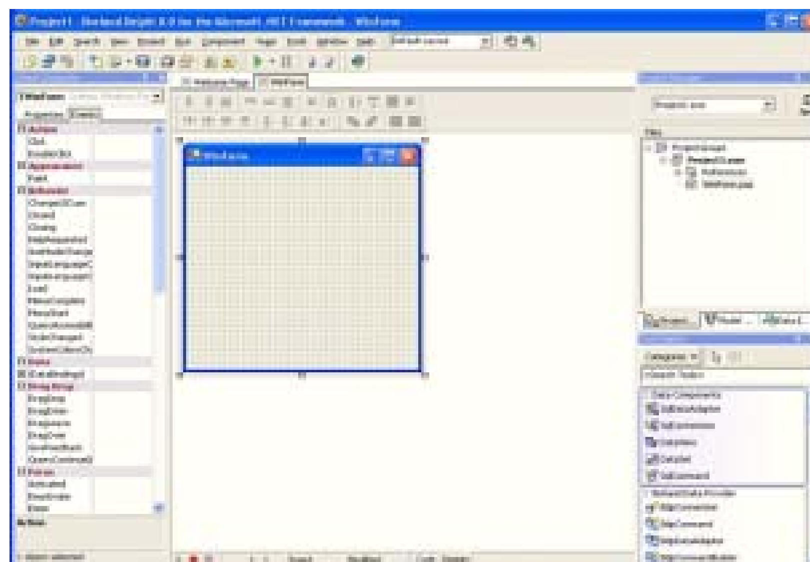
An even more useful test of the web service will be to actually import and use it - in another Delphi for .NET application, of course, which is done in the second part of this article.

So close the browser, close the project (save everything if you haven't done that before), and prepare to start another Delphi for .NET project.

Web Services Client

In this second part of this article, I want to show you how Delphi for .NET can import and use (also called consume) web services. You can use the example web service that was made in the first part of this article, or use another web service available on the web. For your convenience (if you didn't build the web service from the first part), I have made some web services available on the internet, including the Celsius conversion web service itself, so you can still play along if you want.

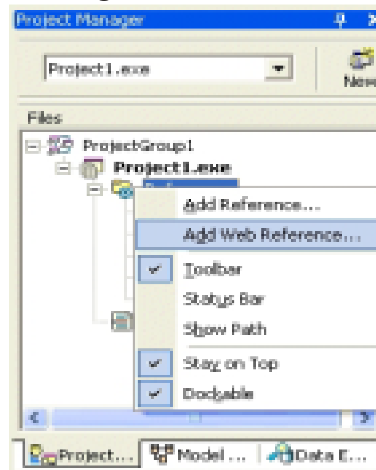
Anyway, start Delphi for .NET and build a new project by doing File | New - Windows Form Application. The result is a new project with an empty form that can be saved where you want, and looks as follows:



I assume you know how to place a control on a form (although this may feel a little bit different if you come from a Delphi/C++Builder background and are using Delphi for .NET for the first time). Before we add visual components to the form, however, I first want to import the web service and add it to our project (so at least we know what we should build upon).

Add Web Reference

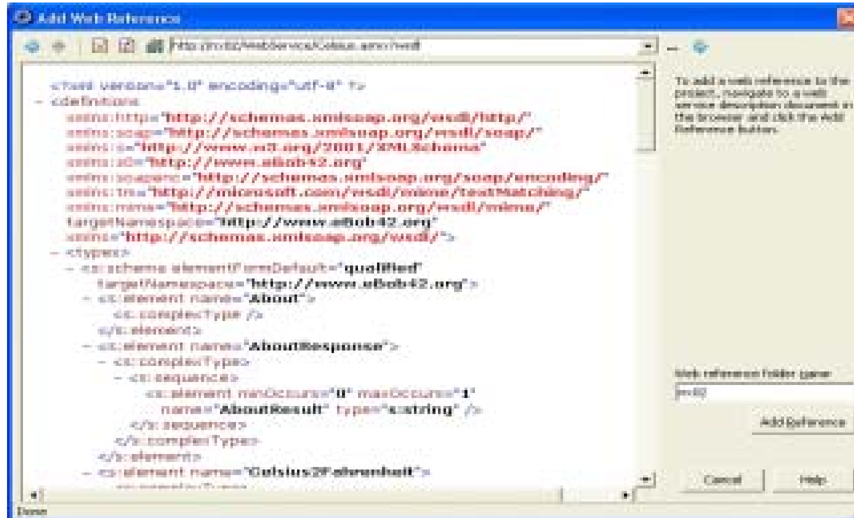
In order to import a web server, we have to add a web reference (that's what it's called in Delphi for .NET), using the Project | Add Web Reference dialog.



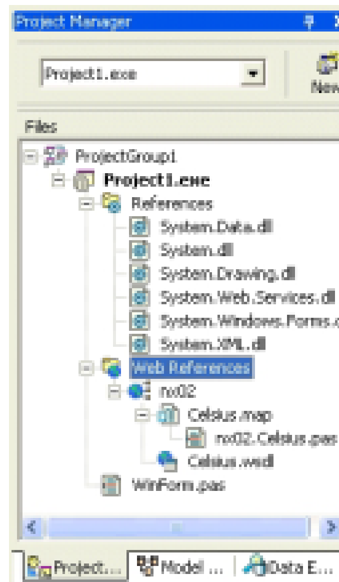
This dialog consists of a URL editbox and browser window, and starts with the Borland Delphi for .NET Universal UDDI Browser to UDDI Services - in case you want to import a web service from one of the available UDDI directories.

Although some of my web services are registered with XMethods, I want to use the Celsius web service that was written in the first part of this article (and for which I know the URL, but which isn't registered with a UDDI registry, yet). In case you have the Celsius web service available on your local machine, the URL to enter in the editbox is probably similar to http://localhost/eBob42/Celsius.asmx, otherwise you can use the version that I've deployed on the internet as http://www.eBob42.com/cgi-bin/Celsius42.asmx. In both cases, the dialog should now show the information of the Celsius web service, as can be seen below:

The Add Reference button is still disabled, and that's because this button can only use a formal WSDL (Web Service Description Language) definition of a web service. For that, click on the Service Description link to get an enabled Add Reference button.



Now we can click on the Add Reference button, which will take a few seconds to analyze the WSDL and produce additional files that are added to the project. Since I've imported the Celsius web service from my nx02 example, we get a new node nx02, with subnodes for the Celsius.wsdl file and a Celsius.map file that refers to the imported web service itself. The import code is placed in nx02.Celsius.pas, where the class Celsius is defined in the nx02.Celsius namespace.



The imported source code is interesting to view, but the only interesting part is the class TCelsius with a default constructor, and the methods About, Celsius2Fahrenheit and Fahrenheit2Celsius that we can use very easily.

Client Form

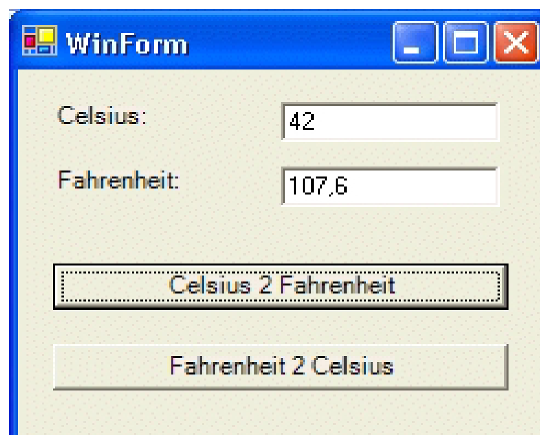
Time to start some visual programming on the WinForm. Drop two Label, two TextBox, and two Button components on the form. Call the first TextBox tbCelsius and the second one tbFahrenheit, and call the first Button btnCelsius2Fahrenheit and the second one btnFahrenheit2Celsius. Now, put "Celsius: " in the Text property of the first Label, and "Fahrenheit: " in the Text property of the second Label. Similarly, put "Celsius 2 Fahrenheit" in the Text property of the first Button, and "Fahrenheit 2 Celsius" in the Text property of the second Button. If you want everything to appear clean, you can also clear the Text property of the two TextBoxes.

Time to write some event handling code for the two Buttons. First, add both the SysUtils and nx02.Celsius units to the uses clause. Now we can implement the two event handlers. In both cases, we need to create a new instance of the TCelsius class, and then call the required methods.

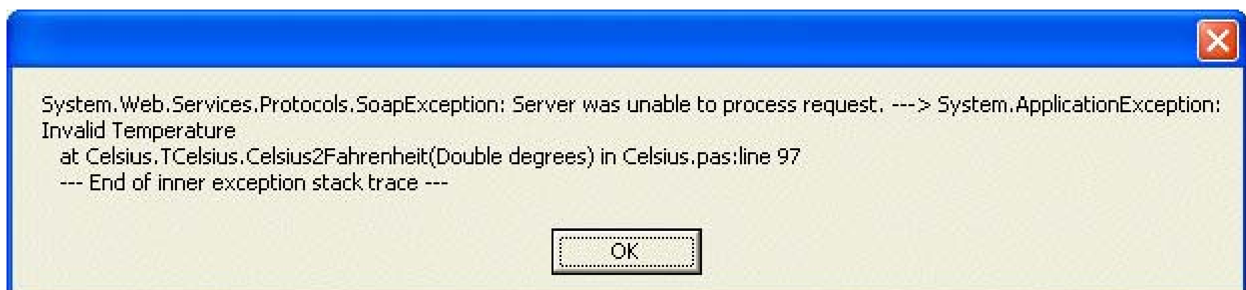
```
procedure TWinForm.btnFahrenheit2Celsius_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: nx02.Celsius.TCelsius;
begin
  C2F := nx02.Celsius.TCelsius.Create;
  tbCelsius.Text := FloatToStr(
    C2F.Fahrenheit2Celsius(StrToFloatDef(tbFahrenheit.Text,0)))
end;

procedure TWinForm.btnCelsius2Fahrenheit_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: TCelsius;
begin
  C2F := TCelsius.Create;
  tbFahrenheit.Text := FloatToStr(
    C2F.Celsius2Fahrenheit(StrToFloatDef(tbCelsius.Text,0)))
end;
```

Note that you should probably want to place all code inside a try-catch block, since creating an instance of the web service can fail (if you cannot connect to it for some reason), but even if you have created an instance, the call to Celsius2Fahrenheit or Fahrenheit2Celsius can also fail or throw an exception - for example when trying to convert a temperature which is below the absolute zero. You can experiment and discover this in practice for yourself.

Now, do File | Save All to make sure everything is saved, compile and run the project which will show the Celsius Client. As a quick demo, I've entered 42 in the tbCelsius TextBox and clicked on the bntCelsius2Fahrenheit button:



When entering a value of -400 in the TextBox for Celsius, and then trying to convert from Celsius to Fahrenheit will result in an exception being thrown by the web service, as shown below:



A similar error dialog (with a longer message) will be shown if the web service isn't available, and hence the call to create an instance of the Celsius.localhost.Celsius web service constructor will fail. Using try-except will make sure that the client application can intercept the exception and optionally present some more useful information to the endusers. This can be done with a code construct as follows:

```
procedure TWinForm.btnFahrenheit2Celsius_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: nx02.Celsius.TCelsius;
begin
  try
    C2F := nx02.Celsius.TCelsius.Create;
    tbCelsius.Text := FloatToStr(
      C2F.Fahrenheit2Celsius(StrToFloatDef(tbFahrenheit.Text,0)))
  except
    on E: Exception do
      MessageBox.Show(E.Message)
  end
end;

procedure TWinForm.btnCelsius2Fahrenheit_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: TCelsius;
begin
  try
    C2F := TCelsius.Create;
    tbFahrenheit.Text := FloatToStr(
      C2F.Celsius2Fahrenheit(StrToFloatDef(tbCelsius.Text,0)))
  except
    on E: Exception do
      MessageBox.Show(E.Message)
  end
end;
```

Obviously, it would be a good idea to give a more user-friendly error message in the except-clause, but I leave that as an exercise for the reader.

Summary

In this article I've used Delphi for .NET - Borland's new IDE for Microsoft's .NET Framework - to build, implement, deploy and consume an ASP.NET web service. We've seen numerous IDE features that support web service developers and consumers while working with ASP.NET web services in Delphi for .NET.

Bob Swart (aka Dr Bob - www.drbob42.net) is a software developer, author, trainer, consultant and webmaster for his own one-man company, Bob Swart Training & Consultancy in Helmond, The Netherlands. He writes for numerous computing magazines as well as his own training material, and is also webmaster to the group.