

Delphi 2005's Web Services

by Craig Murphy

Delphi 2005 allows us to create web services for .NET and for Win32. It also allows us to create web service consumers using WinForms, VCL for .NET and Win32. This article explores each of those and will give you a high-level overview of Delphi 2005's web service capabilities.

Introduction

Web services are all around us. What is interesting about web services is the fact that they have been around for a long time. For example, simply using a browser to look at a web page on a server could be construed as running a web service. This is more evident if the web page is actually a VBScript or JavaScript ASP or a PHP script, etc. More so if the client's browser passes in query strings that instruct the script to do something unique for that particular client.

However, as professional developers, we have probably steered away from scripting languages, favouring the compiled approach or, more recently, the just-in-time (JIT) approach. We probably cited security and performance as reasons not to adopt web service architectures.

Web Services in .NET

Delphi 2005, like Delphi 8, brings with it access to .NET's underlying web service architecture. That architecture makes building web service oriented applications almost as easy as it is to build a Windows application using Win32/VCL controls: drag and drop. Indeed, one of the design goals for .NET was to bring web development into the same easy-to-use environment that desktop had undergone with the introduction of great IDEs like Delphi 1 and, to an extent, Visual Basic. That said, Visual Studio.net has made building web service based applications remarkably easy: now Delphi 2005 offers us that power tool!

If we wish to build and test web services locally, we must have a web server running. I will be using Internet Information Services (IIS), although you are free to choose your own.

The first step is to create a new Delphi for .NET project, an ASP.NET web service to be more precise. Figure 1 presents Delphi 2005's New Items dialog.

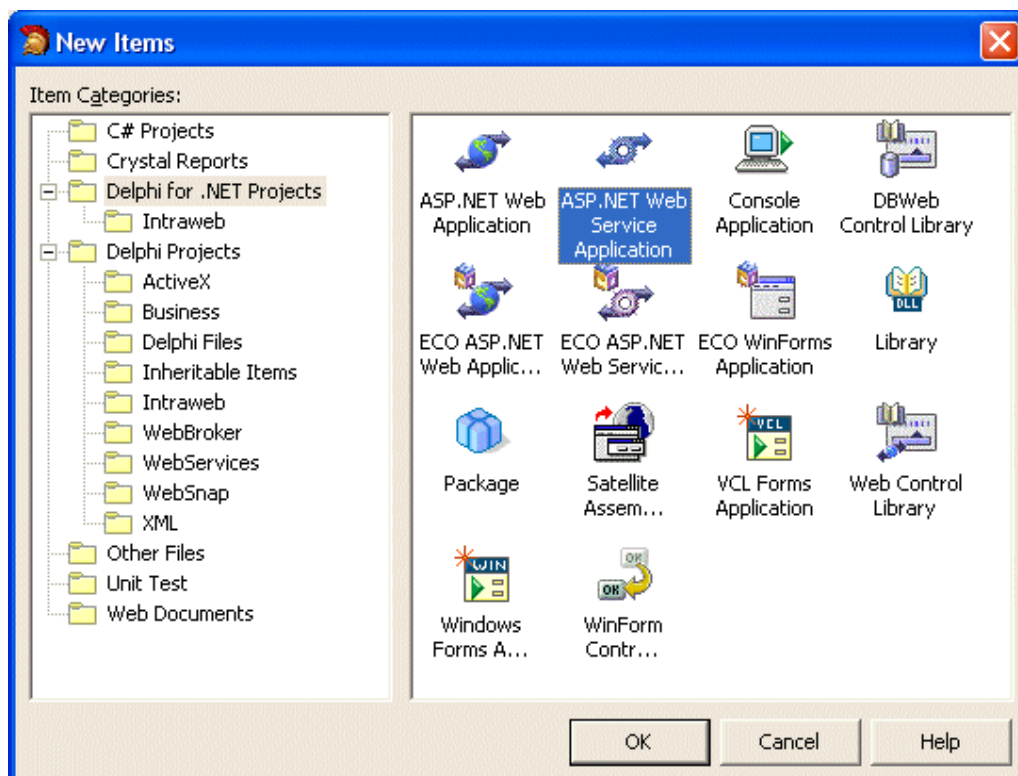


Figure 1: Creating a web service using Delphi 2005

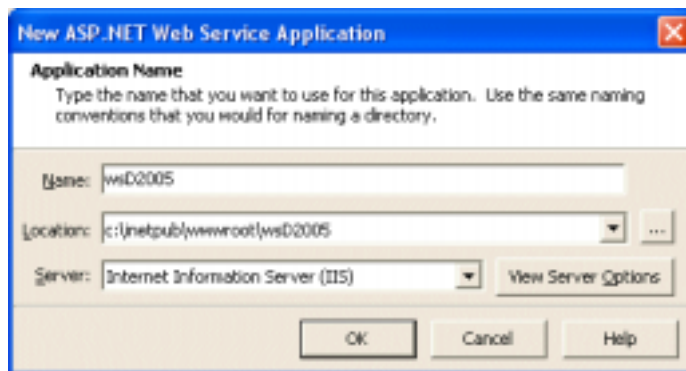


Figure 2: Naming the web service and choosing a web server

After clicking on OK, Delphi 2005 then goes off to create the .NET web service. In doing so it creates a sample web method that demonstrates how to write methods that can be invoked using the web service. This is really rather useful as the first hurdle we are likely to encounter is confirming that the web service actually works. Listing 1 presents the source code for the sample web service: I used Delphi 2005's Rename refactoring to change the name of the web service and have stripped out some comments to preserve space.

```

unit wsGreeting;

interface
uses
  System.Collections, System.ComponentModel,
  System.Data, System.Diagnostics, System.Web,
  System.Web.Services;

type
  TWebServiceGreeting = class(System.Web.Services.WebService)
  {$REGION 'Designer Managed Code'}
  strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    components: IContainer;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
  {$ENDREGION}
  strict protected
    procedure Dispose(disposing: boolean); override;
  private
    { Private Declarations }
  public
    constructor Create;
    [WebMethod]
    function HelloWorld: string;
  end;

implementation

{$REGION 'Designer Managed Code'}
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWebServiceGreeting.InitializeComponent;
begin
end;
{$ENDREGION}

constructor TWebServiceGreeting.Create;
begin
  inherited;
  InitializeComponent;
end;

```

```

procedure TWebServiceGreeting.Dispose(disposing: boolean);
begin
    if disposing and (components <> nil) then
        components.Dispose;
    inherited Dispose(disposing);
end;

function TWebServiceGreeting.HelloWorld: string;
begin
    Result := 'Hello World';
end;

end.

```

Listing 1: Delphi 2005 creates a sample web service automatically

Invoking the web service is the simple matter of pointing your browser to: <http://localhost/wsD2005/wsGreeting.asmx> at which point your Figure 3 should look familiar.



Figure 3: Invoking our first Delphi 2005 web service

The ASP.NET run-time recognises that wsGreeting.asmx is a web service and presents us with the web service's list of operations, in this case a method called HelloWorld. Clicking on the Service Description link leads us to the Web Service Description Language (WSDL) for this particular web service. Thankfully we do not have to worry about the WSDL as Delphi and other ASP.NET tools take care of this for us.

Clicking on Hello World will present us with an opportunity to invoke the web service. Figure 4 presents the screen that we should expect having clicked on Hello World.



Figure 4: Preparing to invoke the wsGreeting web service

The routine that created Figure 4 is smart enough to know about the Hello World method. For instance, it knows that it takes no parameters, hence it has offered us the Invoke button. If a web service takes parameters, these are specified in the WSDL document. Thus any process that is said to consume a web service can use the WSDL to determine what parameters to pass into the web service.

In fact, it is the WSDL that helped create the SOAP request and response shown in Figure 4. In days gone by we, as developers, had to create these responses manually: a tiresome and error-prone task.

Hello World is the simplest possible web service that we could write, but it is enough to convince us that the process works from end-to-end? Clicking on Invoke reveals Figure 5.

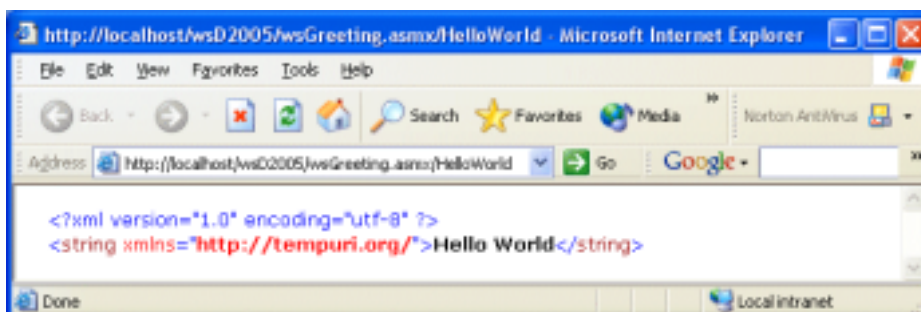


Figure 5: Hello World via a Delphi 2005 web service

Having seen how easy it is to create web services and test them via a browser window, how easy is it to create an application that consumes (uses) the web service that we just created? Well, that is what we will look at next. Delphi 2005 offers us two approaches to web service invocation: using WinForms or using the VCL for .net.

Consuming a Web Service using WinForms

First things first: we need to create a new Windows Forms Application. This can be achieved via the File→New menu as shown in Figure 6.

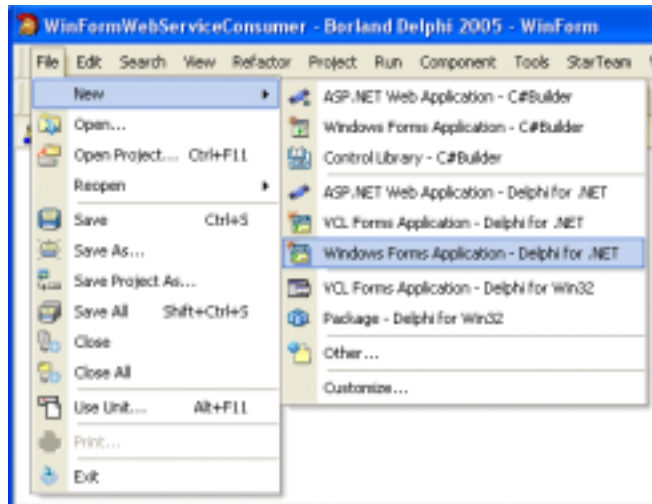


Figure 6: Creating a Delphi WinForms application

Once Delphi 2005 has created the WinForm application, we need to lay out the WinForm as shown in Figure 7. We need a couple of Labels, two TextBoxes and a Button component.



Figure 7: The WinForm Web Service Consumer User Interface

The next step is to add the web service that we wish to invoke to the current project. This can be achieved by right-clicking on the Project Manager dock window then choosing Add Web Reference, as shown in Figure 8. Alternatively, the Project→Add Web Reference menu option achieves the same.

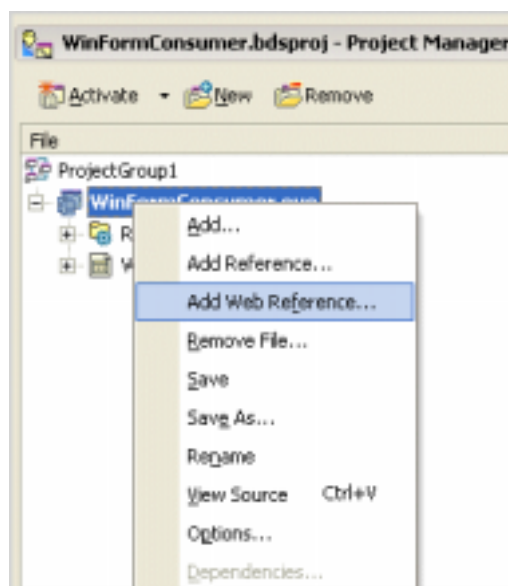


Figure 8: Adding the Web Service to the project

Adding a web reference requires us to provide the URL of the web service that we wish to use. In this instance, it is this URL: <http://localhost/wsD2005/wsGreeting.asmx>. Figure 9 presents a screenshot of where we are so far.

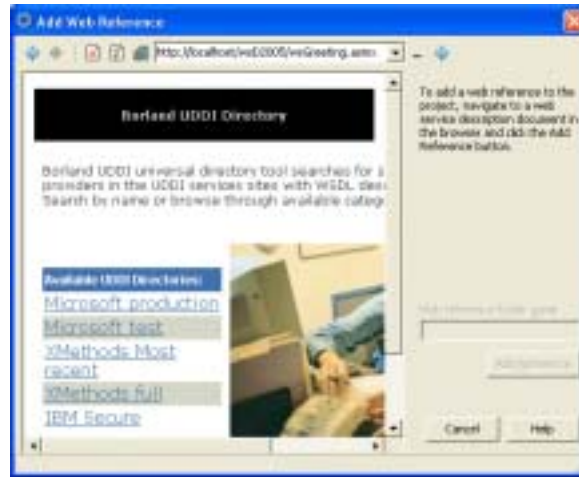


Figure 9: Specifying which web service we wish to add

At this point, your firewall might raise an alert! Clicking the Go [blue] arrow invokes disco.exe: this is a .NET executable that is capable of discovering web services from their URL. Similarly, wsdl.exe should also trigger your firewall as it goes off in search of the WSDL for this web service. Assuming that we allow disco.exe and wsdl.exe to do their stuff, Figure 10 should look familiar.

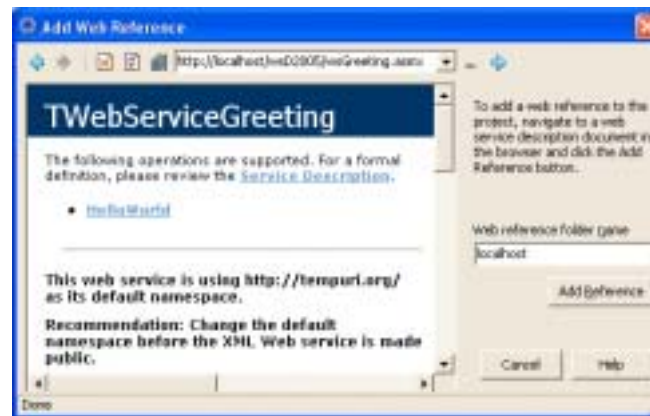


Figure 10: Adding a web reference confirms the operation is available

All that is required now is to click on Add Reference.

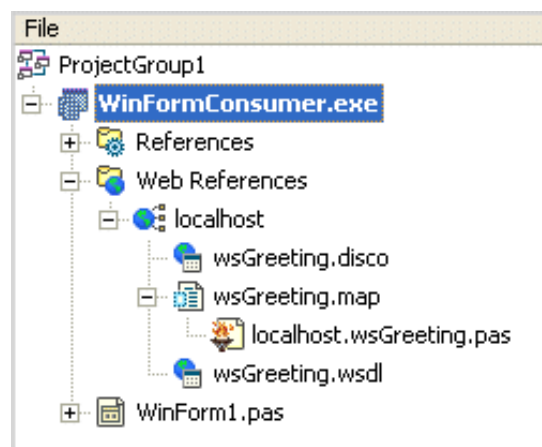


Figure 11: The Project Manager after adding a web reference

Now that we have added a reference to the web service, Delphi has created a new unit: localhost.wsGreeting. It contains a proxy class that is used to invoke the web service(s). Creating a class proxy for the web service means that we can enjoy all the benefits of design-time type checking: something would be sorely lacking if we tried to invoke this web service using the raw SOAP request/response that we saw earlier.

I have included the contents of this unit in Listing 2 for the sake of completeness – I would not expect to make any serious use of this unit other than for the name of the proxy class and the method(s) it surfaces.

```
unit localhost.wsGreeting;

interface

uses System.Diagnostics,
    System.Xml.Serialization,
    System.Web.Services.Protocols,
    System.ComponentModel,
    System.Web.Services, System.Web.Services.Description;

type
  [System.Diagnostics.DebuggerStepThroughAttribute]
  [System.ComponentModel.DesignerCategoryAttribute('code')]
  [System.Web.Services.WebServiceBindingAttribute(Name='TWebServiceGreetingS' +
  'oap', Namespace='http://tempuri.org/')]
  TWebServiceGreeting = class(System.Web.Services.Protocols.SoapHttpClientProtocol)
  public
    constructor Create;
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute('http://tempu' +
    'ri.org>HelloWorld', RequestNamespace='http://tempuri.org/', ResponseNamespace='h' +
    'ttp://tempuri.org/', Use=System.Web.Services.Description.SoapBindingUse.Literal,
    ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    function HelloWorld(strGreeting: string): string;
    function BeginHelloWorld(strGreeting: string; callback: System.AsyncCallback;
    asyncState: System.Object): System.IAsyncResult;
    function EndHelloWorld(asyncResult: System.IAsyncResult): string;
  end;

implementation

{$AUTOBOX ON}
{$HINTS OFF}
{$WARNINGS OFF}

constructor TWebServiceGreeting.Create;
begin
  inherited Create;
  Self.Url := 'http://localhost/wsD2005/wsGreeting.asmx';
end;

function TWebServiceGreeting.HelloWorld(strGreeting: string): string;
type
  TArrayOfSystem_Object = array of System.Object;
var
  results: TArrayOfSystem_Object;
begin
  results := Self.Invoke('HelloWorld', TArrayOfSystem_Object.Create(strGreeting));
  Result := (string(results[0]));
end;

function TWebServiceGreeting.BeginHelloWorld(strGreeting: string; callback:
  System.AsyncCallback;
  asyncState: System.Object): System.IAsyncResult;
type
  TArrayOfSystem_Object = array of System.Object;
begin
  Result := Self.BeginInvoke('HelloWorld', TArrayOfSystem_Object.Create(strGreeting),
  callback, asyncState);
end;

function TWebServiceGreeting.EndHelloWorld(asyncResult: System.IAsyncResult): string;
type
  TArrayOfSystem_Object = array of System.Object;
var
  results: TArrayOfSystem_Object;
begin
  results := Self.EndInvoke(asyncResult);
  Result := (string(results[0]));
end;

end.
```

Listing 2: localhost.wsGreeting – the proxy class for our web service

Once we have added localhost.wsGreeting to the uses clause, we are able to make use of the web service.

The code behind the button Invoke Web Service is surprisingly short. Listing 3 presents all the code that is required to invoke the wsGreeting web service. Even using strResponse, the code is surprisingly short and easy to understand.

```
procedure TWinForm1.btnInvoke_Click(sender: System.Object; e: System.EventArgs);  
var myWebService : TWebServiceGreeting;  
    strResponse : string;  
begin  
    myWebService := TWebServiceGreeting.Create;  
  
    strResponse := myWebService.HelloWorld( tbGreeting.Text );  
  
    tbResponse.Text := strResponse;  
end;
```

Listing 3: Invoking the web service is really this easy!

Taking a further look at Listing 3, the line that is highlighted in bold does all the hard work for us. If you have a firewall running, it is this line that actually instantiates a connection with the [remote] web service, hence your firewall should attempt to block the connection. Obviously this presents us with a few design issues and we may have to involve our network administrators in a discussion about which web services are considered allowable.

Consuming a Web Service using the VCL

It is possible to consume the same web service using the VCL for .NET. The process is identical, except that we create a VCL Forms Application instead of a WinForms Application. All the code is the same as that for the WinForms consumer at which we have just looked.

Web Services in Win32

Building and consuming web services using Delphi 2005 in a Win32 environment is not very different from doing the same using Delphi 6 or 7.

Win32 Server

File → New → Other → Delphi Projects → Web Services

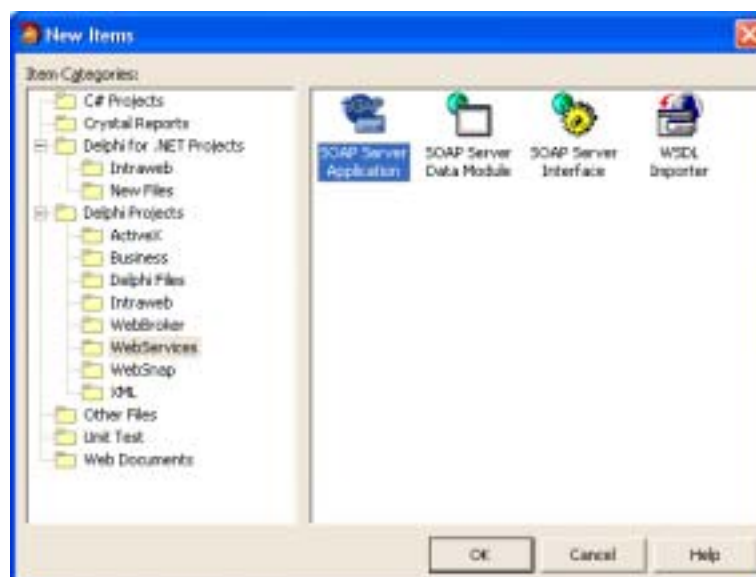


Figure 12: Creating a new SOAP Server Application

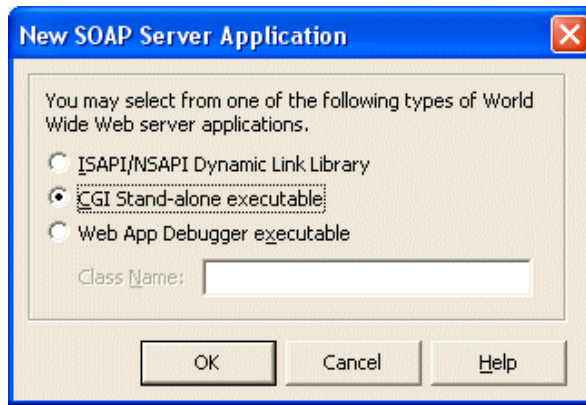


Figure 13: Selecting a cgi executable

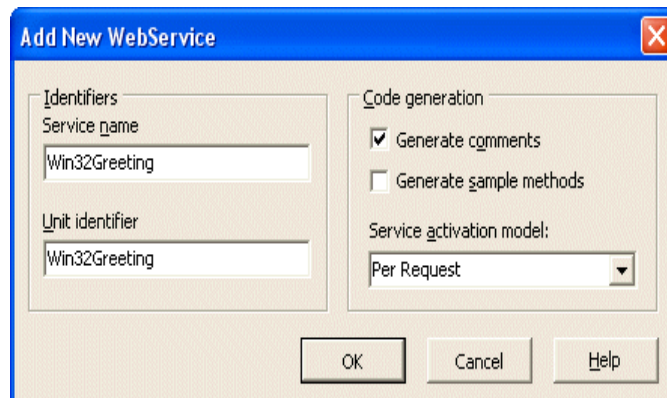


Figure 14: Specifying the web service name

```

{ Invokable interface IWin32Greeting }

unit Win32GreetingIntf;

interface

uses InvokeRegistry, Types, XSBuiltIns;

type

  { Invokable interfaces must derive from IInvokable }
  IWin32Greeting = interface(IInvokable)
  ['{68885429-E463-4F6E-8918-F96FF272056F}']

  { Methods of Invokable interface must not use the default }
  { calling convention; stdcall is recommended }
  function echoGreeting(const strGreeting: String): String; stdcall;
  end;

implementation

initialization
  { Invokable interfaces must be registered }
  InvRegistry.RegisterInterface(TypeInfo(IWin32Greeting));

end.

```

Listing 4: The Win32 web service interface

```

{ Invokable implementation File for TWin32Greeting which implements IWin32Greeting }

unit Win32GreetingImpl;

interface

uses InvokeRegistry, Types, XSBuiltIns, Win32GreetingIntf;

type

  { TWin32Greeting }
  TWin32Greeting = class(TInvokableClass, IWin32Greeting)
  public
    function echoGreeting(const strGreeting: String): String; stdcall;
  end;

implementation

{ TWin32Greeting }

function TWin32Greeting.echoGreeting(const strGreeting: String): String;
begin
  Result := 'Hello, ' + strGreeting;
end;

initialization
  { Invokable classes must be registered }
  InvRegistry.RegisterInvokableClass(TWin32Greeting);

end.

```

Listing 5: The Win32 web service implementation

Compiling the Win32 project that includes Listing 4 and Listing 5 will create an exe that can be used to provide not just the web service implementation/method, but the WSDL too. But first we have to copy the exe into our web server's cgi scripting area: I have chosen to copy it to a directory called D2005_W32.

The first thing we should do is test that we can see the WSDL. This can be achieved by going to the following URL: http://localhost/D2005_W32/W32GREETING.EXE/wsdl (assuming that the exe is called W32GREETING!) If all goes to plan, Figure 15 should look familiar:



Figure 15: It works. Win32 WSDL is available

Now that we know how to build the server-side web service, it is time to look at creating a client that consumes it.

Win32 Client

The Win32 client will be nothing more than a regular VCL Forms Application: File → New → Other, as depicted by Figure 16.

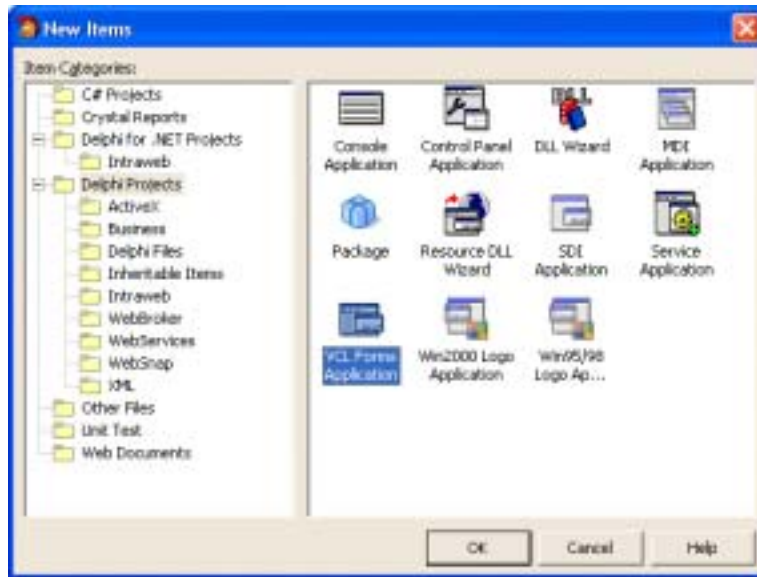


Figure 16: Creating a VCL Forms Application

The next thing we need to do is import the WSDL that the server-side web service produces. If you recall Figure 15, there was a [WSDL] link beside the IWin32Greeting PortType – this link produces WSDL that describes the web service.

Importing the WSDL makes use of the WSDL Importer: File → New → Other → Delphi Projects → Web Services.

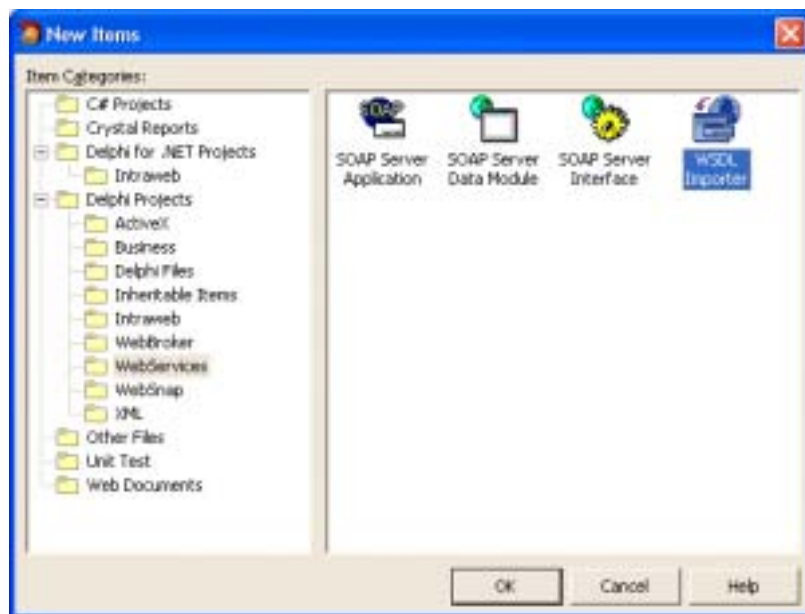


Figure 17: Importing WSDL is easy using the WSDL Importer

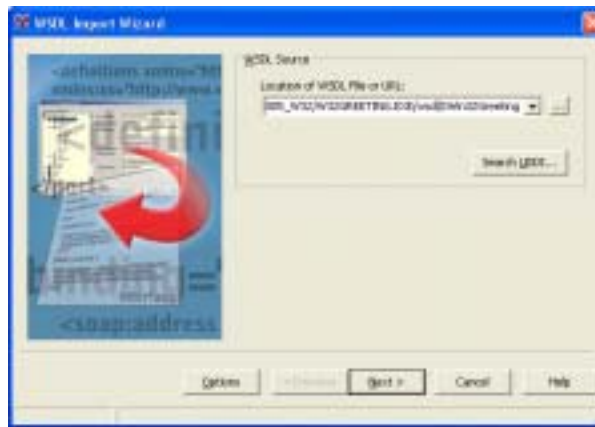


Figure 18: Specifying the WSDL that we wish to import

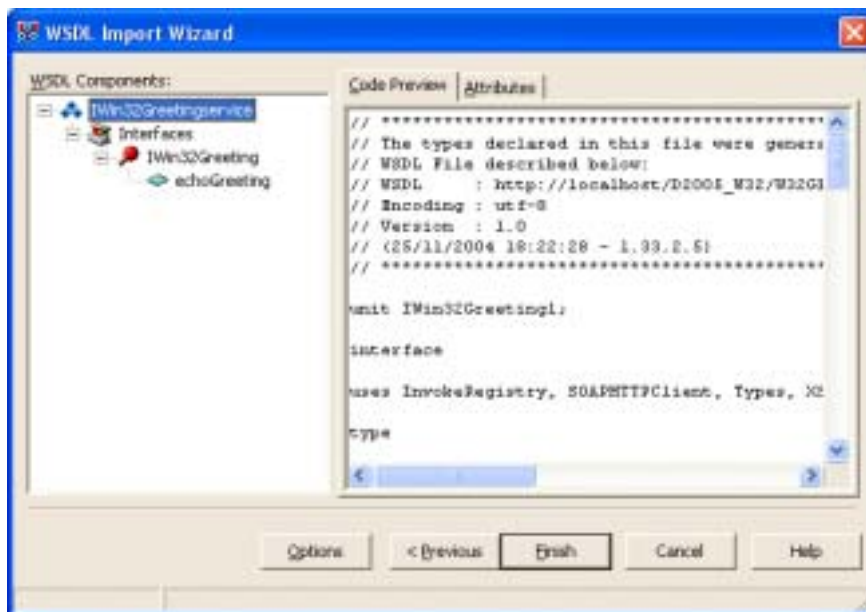


Figure 19: Importing the WSDL reveals the web service interfaces

The WSDL importer will then create a new unit, in this case IWin32Greeting1. It contains an interface representing the web service:

```
IWin32Greeting = interface(IInvokable)
[ '{888C8498-104E-967D-E3DE-DC3F5A09BD0}' ]
function echoGreeting(const strGreeting: WideString): WideString; stdcall;
end;
```

Now, if we drop a couple of TLabel components, a TButton, a couple of TEdit components and a THTTPRIO component onto the main form we can then test the web service. The arrangement of the components should be similar to the WinForms example we saw earlier. The code behind the button click should look like Listing 6.

```
procedure TForm1.btnInvokeClick(Sender: TObject);
var myWebService : IWin32Greeting;
begin
myWebService := HTTPRIO as IWin32Greeting;

edtResponse.Text := myWebService.echoGreeting( edtGreeting.Text );
end;
```

Listing 6: Invoking the Win32 web service using a Win32 client applications

Now, if we invoke the web service using the text "DELPHI 2005 ROCKS!" Figure 20 should look gratifyingly familiar:



Figure 20: Putting it all together

Summary

This article has provided a high-level overview of Delphi 2005's web service capabilities. A future article will look at building a more functional and more practical web service based application.

Web services should be platform and language-agnostic. In other words, we really should not care what language a web service is written in, nor should we care about which language is consuming a web service. In reality there are some interoperability issues, so it is still worth proceeding into the web services arena with care.



Craig is an author, developer, speaker, project manager, Certified ScrumMaster and Microsoft Most Valuable Professional (XML Web Services). He specialises in all things XML, particularly SOAP and XSLT. Craig is evangelical about .NET, C#, Test-Driven Development, Extreme Programming, agile methods and Scrum. He can be reached via e-mail at: bug@craigmurphy.com, or via his web site: <http://www.craigmurphy.com> (where you can also find the source code and PowerPoint files for all of Craig's articles, reviews and presentations).

Developer-friendly web hosting and dedicated servers with great support

Are you fed up with explaining what you need to your web host? Wish they understood about .NET, ISAPI, CGI and XML? Wish you could get some real technical support?

At TDMWeb we know about software development: we also publish *The Delphi Magazine*, and deal with developers all day long!

So talk to us and you're talking with friends. We even have a special Windows Developer package designed just for the development and testing of web applications.

From simple shared hosting to your own dedicated server (complete with our Total Management service if you need it), we can do it all.

- Windows and Linux hosting and dedicated servers.
- ISAPI, ASP, ASP.NET, CGI, PHP, Perl, SSI, MySQL, Access, SQL Server, SOAP, XML, XSLT and more.
- Spam and virus protection.
- Excellent prices, great support!

Full package details at www.TDMWeb.com
Call +44 (0)870 740 7610 or Email info@tdmweb.com

TDMWeb
www.tdmweb.com