

Dr Bob's Prescriptions

Nº. 2: SOAP Bubbles

This is the second appearance of Dr.Bob's Prescriptions page, where Bob Swart shares some tips, tricks or general sensible steps to take with regards to internet technology and Delphi. This time, he focuses a bit on Web Services and SOAP - the Simple Object Access Protocol, supported in Delphi 6 Enterprise.

There are a number of important things you have to know when working with Delphi 6 and producing Web Services. The biggest problem we found was a memory leak in the Web Service consumers that could be as much as 12KBytes per remote method invocation (this memory was never freed, until you closed down the client application - no memory leaks or hogs were reported on the Web Service engine side).

Delphi 6 Update 1

It's important to make sure you have downloaded and installed the Delphi 6 Update patch #1 (the new one, that is, of September 27th 2001), especially if you want your Delphi 6 generated Web Service engines to be used by other environments (like Visual Studio .NET),. For more information about this Delphi 6 Update patch #1, including a list of fixes, see <http://community.borland.com/article/0,1410,27800,00.html>.

In addition to a number of significant improvements to Delphi's SOAP (Simple Object Access Protocol) implementation, this Update Pack offers a number of important modifications and addresses a number of issues discovered in the initial release of this product. The above URL lists a summary of Update Pack improvements. SOAP-specific improvements are listed separately. For details on modifications and improvements to these and other features, see the file D601FIXES.html, installed to your Delphi 6 root directory (after applying the patch, that is).

TargetNamespace

The TWSDLHTMLPublish component now exposes a TargetNamespace property. The value defaults to <http://www.borland.com/soapServices>. With this new property in place, we strongly recommend that you update the TargetNamespace of your Web Services before deployment.

Note that the original version of Delphi 6 generates the following SOAP namespace:

```
xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap"
```

while this should end with a trailing /, i.e.

```
xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
```

That's an important change that you should make in your WSDL that Delphi 6 creates, in order to make your Delphi 6 Web Services work with the Microsoft SOAP Toolkit client. This means that you should generate WSDL for your Delphi 6 Web Service, modify it, and distribute it (instead of using the dynamic WSDL again).

WSDLADMIN.INI

When I uploaded the first Web Services that I made on my web server, I noticed that as soon as I requested the WSDL, some .INI files were automatically created with names like xxx_WSDLADMIN.INI. I never really paid attention to them until I enhanced the interface of one of my Web Services, re-uploaded the Web Service, but found myself unable to connect and call this new method.

It appears that the Publish code takes a different path if the .INI file is present because it assumes that you've administered your Service and potentially added, removed or modified one or more service endpoints. It seems that the Delphi 6 generated Web Services generate the .INI files even upon simple WSDL requests, and Borland even worked on a fix - one which unfortunately didn't make it into the official Update patch #1 for Delphi 6.

The scenario under which this shows up is if you deploy a Service and access it; and then add a new interface. The endpoint entry for the new interface will be missing; we'll assume you've administered the service and made the service unavailable at its default endpoint. The easy fix for now is to delete the .INI file if you add a new interface to a deployed service.

While on the subject of WSDLADMIN.INI, I should mention one more thing: the code fails to check the AdminEnabled flag before allowing administration.

Delphi 6 eXtreme Toy: Invokable Wizard

Registered Borland Delphi 6 Enterprise users can download an eXtreme Toy called the Invokamatic Wizard for Web Services. This wizard makes it possible to create a new web service with Delphi 6 Enterprise in about 60 seconds (depending on how fast you can implement the logic for the service, of course). If the logic already exists, you should be able to have the web service available to the internet in less than five minutes. The invokable.zip archive that you can download contains the following files:

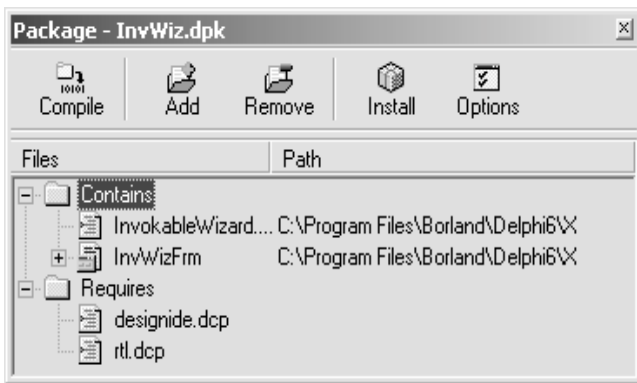
```
2001-10-17 15:44 <DIR>
INVOKABLE
2001-05-21 02:06 872
InvokableWizard.dcr
2001-06-03 02:59 12.189
InvokableWizard.pas
2001-06-03 03:01 362
InvWiz.cfg
2001-06-03 03:04 3.206
InvWiz.dcu
```

2001-06-03	03:01	1.537
InvWiz.dof		
2001-06-03	03:00	1.599
InvWiz.dpk		
2001-05-25	20:39	1.692
InvWiz.res		
2001-06-03	02:47	33
InvWizFrm.ddp		
2001-06-02	08:18	4.790
InvWizFrm.dfm		
2001-06-03	02:47	3.860
InvWizFrm.pas		
2001-06-03	02:56	1.335
readme.txt		

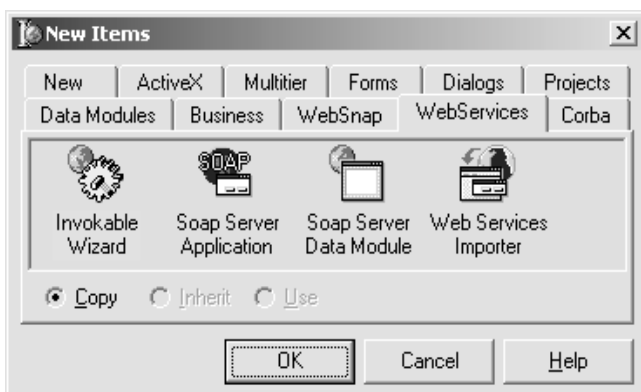
The subdirectory INVOKABLE contains a somewhat older version of the Invokable Wizard (about two weeks older). I have no idea why this is included as well.

2001-05-21	02:06	872
InvokableWizard.dcr		
2001-05-24	01:53	11.738
InvokableWizard.pas		
2001-05-24	01:54	346
InvWiz60.cfg		
2001-05-24	01:54	2.035
InvWiz60.dof		
2001-05-22	03:18	679
InvWiz60.dpk		
2001-05-21	01:03	1.536
InvWiz60.res		
2001-05-24	01:48	51
InvWizFrm.ddp		
2001-05-24	01:48	4.315
InvWizFrm.dfm		
2001-05-24	01:44	2.019
InvWizFrm.pas		

To install the Invokamatic wizard, open InvWiz.dpk in Delphi 6 and click the Install button:



Once you have it installed, the Invokable wizard should show up in the WebServices tab of the object repository.



Invokable Wizard Demo

To create a new web service from scratch, do *File / New / WebServices* and select the SOAP Server Application icon (like we've done before). Select your favourite choice of web server application and save your application in something like SCGI.dpr (and your web module in SWebMod.pas).

Then, do *File / New / WebServices* and select the Invokable Wizard from the Object Repository. This finally shows the Invokable Wizard dialog:



We need to do a few things here. First of all, specify your service name (let's implement the Roman Numbers web service at this time). As you type "RomanNumber", you'll notice the Unit identifier, Interface name, Interface unit (name) and Implementation unit (name) all move along.

You only need to specify the Invokable class type in the combobox, which can be TInterfacedObject or TInvokableClass. The latter is suited for a remote invocation, so make sure to select the TInvokableClass here.



Then click on Generate to generate the two new units: RomanNumerIntf.pas (with the interface definition) and RomanNumberImpl.pas (with the implementation - or rather, the placeholder where you should write your implementation code for the interface). Note that the two units have automatically been added to your project main source file, so you can now take a look at the units themselves and follow their clear instructions.

The unit RomanNumberIntf.pas contains instructions for you to add methods to the interface:

```
{ Invokable interface declaration unit
    for IRomanNumber }

unit RomanNumberIntf;
interface
uses
    Types, XSBuiltIns;
```

```

type
  IRomanNumber = interface(IInvokable)
  ['{A2229FA5-A20F-4AF0-8A8C-
    63B848428AB6}']
  // Declare your invokable logic
  // here using standard
  // Object Pascal code
  // Remember to include a calling
  // convention! (usually
  // stdcall)

end;

implementation
uses
  InvokeRegistry;

initialization
  InvRegistry.RegisterInterface(TypeInfo
    (IRomanNumber), '', '');
end.

```

Unit RomanNumberIntf.pas

You could, for example, add a RomanToInt and an IntToRoman method, to convert Roman numerals into normal integers. After you've defined the interface, you should move to the RomanNumberImpl.pas unit, and read the instructions there:

```

{ Invokable implementation declaration
  unit for TRomanNumber,
  which implements IRomanNumber }

unit RomanNumberImpl;
interface
uses
  RomanNumberIntf, InvokeRegistry;

type
  TRomanNumber =
    class(TInvokableClass,
      IRomanNumber)
    // Make sure you have your
    // invokable logic
    // implemented in
    // IRomanNumber

```

```

// first, then use CodeInsight(tm)
// to fill in this
// implementation
// section by pressing Ctrl+Space,
// marking all the interface
// declarations for IRomanNumber,
// and pressing Enter.
// Once the declarations are
// inserted here, use
// ClassCompletion(tm)
// to write the implementation
// stubs by pressing
// Ctrl+Shift+C

```

```
end;
```

```
implementation
```

```
initialization
```

```
  InvRegistry.RegisterInvokableClass
    (TRomanNumber);
```

```
end.
```

Unit RomanNumberImpl.pas

Again very clear instructions, and you should repeat the definition for the RomanToInt and IntToRoman methods, making sure actually to implement them at this location. Then, just compile and deploy your Web Service application like we've done before (or see <http://www.drbob42.com/soap> for details).

A fully implemented web service that actually publishes and implements RomanToInt and IntToRoman can be found at <http://www.eBob42.com/cgi-bin/Romulan.exe>.

Finally, a little warning: the only support available for the Invokable Wizard is through the Borland public newsgroup borland.public.delphi.webservices.soap.

Bob Swart (aka Dr.Bob - www.drbob42.com) is an independent author, trainer and webmaster using Delphi, C++Builder and Kylix.



He lives in The Netherlands, but loves to shop in London. He's a firm believer in (and addicted to) e-business and spends lots of money on Amazon.com and other e-commerce sites.