

# Dr.Bob's Prescriptions No 3: JBuilder 6 Enterprise and the WebServices Kit for Java

2001 was a good year for JBuilder. Starting at version 4, it has already reached version 6. In other words, it has surpassed C++Builder when it comes to version numbers, and is now equal to Delphi 6. Just how equal JBuilder and Delphi are in other areas will be shown in this article, where I use JBuilder 6 Enterprise with the new WebServices Kit for Java (also from Borland) to connect to (consume) the IEuro WebService that I wrote last year using Delphi 6 Enterprise.

If you don't have JBuilder 6 Enterprise (and I don't blame you - even as loyal customer it's hard to pay the upgrade price twice a year), you can always download the 30-day trial version. Yes, that's 30 days trial, and no longer 60 days like it was with Delphi 6 Enterprise or Kylix 2 Enterprise.



After you've started JBuilder 6, do *File / New Project* to start a new project. Specify EuroClient as the Name of the new project and click on Finish to create the new EuroClient project.

The first thing you need to do now is to start a new application - the web service client (also called consumer). So do *File / New* and select

Application from the Object Gallery. On the first page of the Application Wizard dialog that follows, you can specify the Package (Euroclient) and Class (Application1) for your application (just leave the defaults) and on the next page you can specify some frame class options, such as the Class (Frame1) and Title (I've changed the title from "Frame Title" to "Euro Calculator"). Finally, click on Finish to create the new application and frame.

## Euro WebService

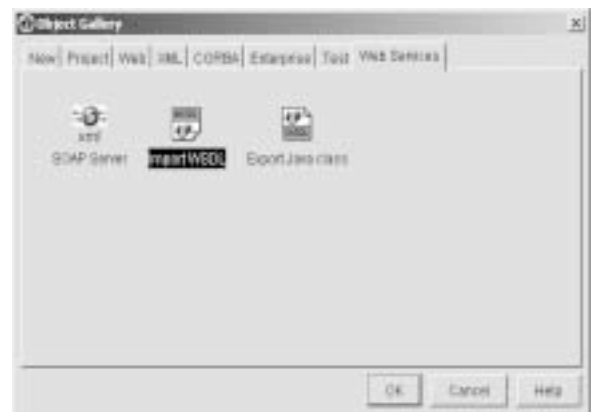
First of all, I hope you've seen (or heard about) the Euro web service that I implemented in Delphi 6 Enterprise as well as Kylix 2 Enterprise some time ago. If not, then that's no problem, since it's available on the web, and easy to use (as I'll show in this article). The IEuro WebService consists of two methods: FromEuro and ToEuro, both with two arguments: Currency and Amount. The first argument, Currency, is a 3-character string and defines the currency the web service will convert from or to Euros. At this time, twelve countries are ready to start using the Euro as of 1-1-2002, so we have twelve currencies: "FIM", "ITL", "NLG", "ESP", "BEF", "ATS", "LUF", "DEM", "FRF", "IEP", "PTE", and "GRO". The second argument, Amount, is a double that contains the amount of money we want to convert from or to Euros.

The Web Service itself is deployed on the internet, and can be reached and consumed from <http://www.eBob42.com/cgi-bin/Euro42.exe/wsdl/IEuro>.

## Importing WSDL

So far so good, but we've only made the skeleton application. We now need to add the IEuro WSDL definition to our project. We can do this in two ways: using an existing WSDL file or using a dynamic WSDL URL.

For the WSDL file, we first have to save the WSDL content we saw in Figure 1, put it inside Euro.wsdl, add it to our project (right-click on the project node and select "Add Files / Packages") and then generate Java classes from it, or we can start the Import WSDL Wizard which can be found in the WebServices tab of the Object Gallery:

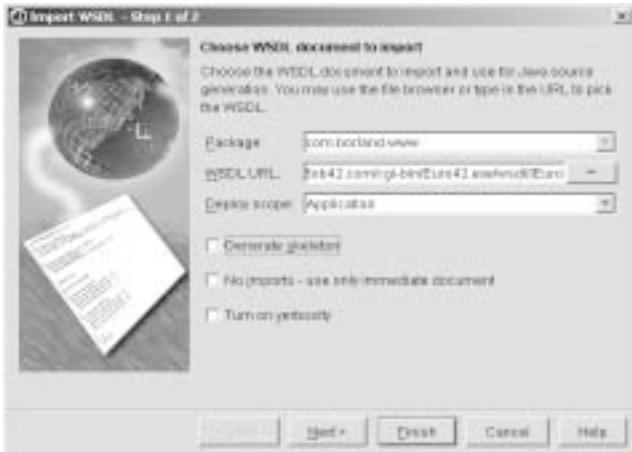


## JBuilder 6 Enterprise

Now that the preparations are finished, let's start JBuilder 6 Enterprise. I assume you have also installed the Borland Web Services Kit for Java, by the way, otherwise you'll find that you can't really work with the WSDL file.

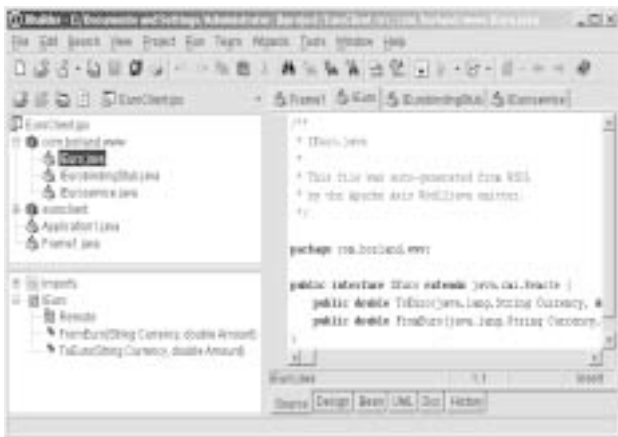
The Import WSDL Wizard is also the tool that we need to use for a dynamic WSDL URL, so let's use this new tool.

As you can see below, we only need to specify the WSDL URL. By default, this is a URL of the form `http://www.eBob42.com/cgi-bin/Euro42.exe/wsdl/IEuro` (for a web service written in Delphi). In case you want to use an existing WSDL file, you can click on the ellipsis and locate the file on your hard disk.



Note that we don't need to generate a skeleton (which would be used to create a Web Service "server"), but only the stub, because we are consuming an existing web service.

After you click on Finish, there will be three new files added to your EuroClient project, placed in the `com.borland.www` package, namely `IEuro.java`, `IEurobindingStub.java` and `IEuroservice.java` (all three generated from the definitions found in the WSDL file).



The generated source file `IEuro.java` contains the definition of the interface `IEuro`, which is derived from `java.rmi.Remote` - meaning that we have a remote invocable interface here - and contains two methods: `FromEuro` and `ToEuro`.

```
/**
 * IEuro.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis Wsdl2java emitter.
 */

package com.borland.www;

public interface IEuro extends java.rmi.Remote
{
    public double FromEuro(java.lang.String
        Currency, double Amount)
        throws java.rmi.RemoteException;
    public double ToEuro(java.lang.String
        Currency, double Amount)
        throws java.rmi.RemoteException;
}
```

The generated source file `IEurobindingStub.java` contains the stub for the `IEuro` binding, meaning the actual class that we can use in our Web Service consumer application, and that will act as the (remote) web service itself, but in fact only consists of a stub that will invoke the remote interface.

## Euro Conversion

Now that the pieces are in place it's time to start using the Web Service. Go back to the frame, click on the Designer tab, and start dropping some Swing components on it.

Personally, I always start by changing the Layout property of the contentPane of the Frame to `XTLayout`. After this, I find it easier to drop and position controls. For this example, we need only five controls: two `javax.swing.JTextField`s, one `javax.swing.JComboBox` and two `javax.swing.JButton`s.

Set the name property of the first `javax.swing.JTextField` to `textFieldCurr`, and set the name property of the second `javax.swing.JTextField` to `textFieldEuro`. Set the text property of the `javax.swing.JTextField`s to `0.0` and you may also want to set their `preferredSize` property to `72,21` in order to make sure they're wide enough to show the converted results.

Set the name property of the `javax.swing.JComboBox` to `comboBoxCurr`. In order to fill the `comboBoxCurr` with the list of twelve possible currencies that can be converted to Euros, we should change the declaration of `comboBoxCurr` in the `Frame1` constructor as follows:

```
String[] EuroCurr =
    {"FIM", "ITL", "NLG", "ESP",
     "BEF", "ATS", "LUF",
     "DEM", "FRF", "IEP",
     "PTE", "GRO"};
JComboBox comboBoxCurr = new
    JComboBox(EuroCurr);
```

And, since I'm from The Netherlands, I've also changed the `selectedIndex` property of `comboBoxCurr` to `2`, so the default choice of the combobox will show `NLG` (but feel free to specify your own default choice here).

Finally, set the name property of the first `javax.swing.JButton` to `buttonFromEuro` and set the name property of the second `javax.swing.JButton` to `buttonToEuro`. And to finish the design-time property setting, change their text properties to "From Euro" and "To Euro" respectively.

## Using IEuro Web Service

Finally, let's write the `ActionPerformed` event handlers for the `buttonFromEuro` and `buttonToEuro` controls. We need to start with two import statements, for `java.net.*` and for `com.borland.www.*` in order to include the classes we need, so that's the following additional code (after the existing import statements):

```
import java.net.*;
import com.borland.www.*;
```

Back to the `ActionPerformed` events. In both event handlers, we need to create binding to the `IEurobindingStub`, which needs a URL pointing to the `http://www.eBob42.com/cgi-bin/Euro42.exe/IEuro` in order to create the binding.

Next, in the `ActionPerformed` event handler for the `buttonFromEuro`, we can use the stub to call the `FromEuro`

method, taking the contents of the jTextFieldEuro control (converted to a Double) as input, and placing the results back in the jTextFieldCurr control.

```
void jButtonFromEuro_actionPerformed
    (ActionEvent e) {
    try {
        IEurobindingStub stub = new
            IEurobindingStub(
                new URL("http://
                    www.eBob42.com/cgi-bin/
                    Euro42.exe/soap/IEuro"));
        jTextFieldCurr.setText(String.valueOf(
            stub.FromEuro(EuroCurr[jComboBoxCurr.
                getSelectedIndex()],
                Double.valueOf(jTextFieldEuro.getText()).
                    doubleValue())));
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

The actionPerformed event handler for the jButtonToEuro is very similar. Instead of calling the FromEuro method, we now call the ToEuro method, and we now take the contents of the jTextFieldCurr control as input, and place the result back in the jTextFieldEuro control.

```
void jButtonToEuro_actionPerformed
    (ActionEvent e) {
    try {
        IEurobindingStub stub = new
            IEurobindingStub(
                new URL("http://
                    www.eBob42.com/cgi-bin/
                    Euro42.exe/soap/IEuro"));
        jTextFieldEuro.setText(String.valueOf(
            stub.ToEuro(EuroCurr[jComboBoxCurr.
                getSelectedIndex()],
                Double.valueOf(jTextFieldCurr.getText()).
                    doubleValue())));
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

After these two event handlers we can save the project and hit F9 to run the application, which - if we didn't make any typing mistakes - will result in the Euro Calculator as shown:



As you can see, you can enter 42 in the left jTextField, select NLG in the jComboBox and click on the "To Euro" button to produce 19.0587690 in the right jTextField (showing the amount in Euros).

## Conclusion

I hope this has shown you that JBuilder 6 Enterprise with the WebServices Kit for Java can easily import (consume) WebServices written in Delphi 6 (and Kylix 2 for that matter). The other way around (producing a WebService in JBuilder that can be used with Delphi 6) is also possible of course, but that's a topic for another day.

*Bob Swart (aka Dr Bob - [www.drbob42.com](http://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 e-commerce sites.*