

Create and consume Web services using a simple and powerful tool

White paper

June 2007



Rational software

SOA made simple with IBM Rational Business Developer software.

Bob Cancilla

System i Software Evangelist

IBM Rational Tools System i/z Strategy/Enablement

Contents

2 Introduction
3 Services and languages
3 The benefits of EGL
12 Consuming a service in EGL
22 EGL is committed to SOA

Introduction

Service-oriented architecture (SOA) has become the catchphrase of many vendors and consultants throughout the IT industry. What is SOA? In a nutshell, SOA is the creation of small reusable components of application systems that isolate logical functionality.

Services are modular components of business logic. Today, many business applications exist as large monolithic programs with data access, business logic and user interfaces embedded in the same programs. These systems often have redundant code that performs the same functions in many programs. SOA provides architecture (or structure) to isolate business functionality into small reusable components that can be assembled into larger systems. This componentization separates presentation from the underlying business logic, improving the stability and maintainability of the modern enterprise.

In addition to improving the maintainability of internally developed systems via modularization, SOA allows you to interact with customer and other vendor systems. Functions or services provided by external parties can be integrated with internally developed systems.

Consider a Web-based order-processing system. After a customer places an order, he or she selects a shipping method. The system can utilize the shipper's remote services to calculate the shipping cost. It also may use an external sales tax calculation service to calculate the applicable taxes for the order. SOA is very similar to traditional modular programming but now extends the reach of services—or modular components—beyond any specific hardware platform, programming language or even geographical location.

Services and languages

A service should focus on the goal of performing a business function. When you look at the concept of creating services that can be invoked from anywhere within your enterprise or via the Internet, communications can add layers of complexity to the use of services.

SOA has been associated with the Java™ language. The reality is that SOA services can be created from virtually any computer language. The key to creating a service is building the interface that enables your service to be invoked anywhere it is required. Many IBM tools—including the IBM WebSphere® Development Studio Client for System i™ servers, IBM WebSphere Development for System z™ servers, as well as the IBM Rational® Application Developer or IBM Rational Software Architect system—can all create services.

If you are new to services and if protocols such as Simple Object Access Protocol (SOAP) or Web Services Description Language (WSDL) are intimidating, you may want to look at simple alternatives. IBM Rational Business Developer Extension software, which contains IBM Rational Enterprise Generation Language (EGL), hides the complexity of middleware and Web protocols, allowing you to concentrate on the business requirement.

The benefits of EGL

A platform-neutral development environment, Rational Business Developer Extension offers both a simple language and powerful tools to automate the creation and consumption of services. It allows you to leverage a single development environment and deploy applications to any supported environment (e.g., IBM AIX®, IBM i5/OS®, Linux®, Microsoft® Windows® or IBM-supported versions of UNIX®). You also can develop Web applications, IBM 5250 code, batch applications and—soon—rich client GUI applications.

Following is a simple example of creating a Web service with EGL, and then consuming that service.

Creating and deploying a service with EGL

To demonstrate the simplicity of EGL and creating a service, let's create a simple currency conversion service. The following example was built using Rational Business Developer Extension installed on top of IBM WebSphere Development Studio Client for System i Advanced Edition software.

To create a service in EGL:

1. Create an EGL Web project.
2. Create a package to contain your services in the EGLSource folder.
3. Create a service.
4. Write the business logic for your service.
5. Generate the EGL Service Binding Library.
6. Test your service using the Web Services Explorer tool.

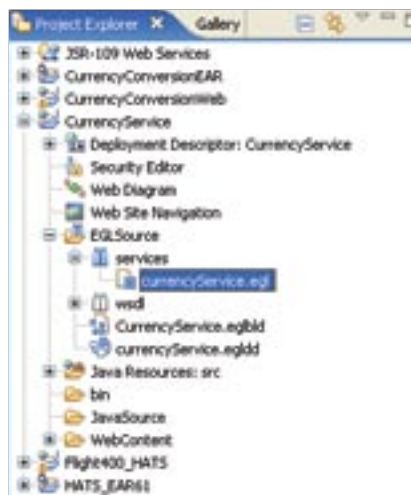
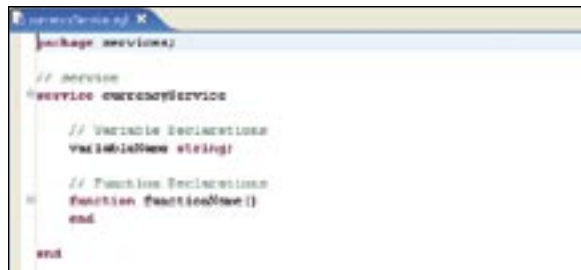


Figure 1: Project name is "currency services."

First, create an EGL dynamic Web project. We named our project “currencyService.” Next, create a package and call it “services” within the EGLSource folder. Select and right-click on the services package; select *New* and then *Service*.

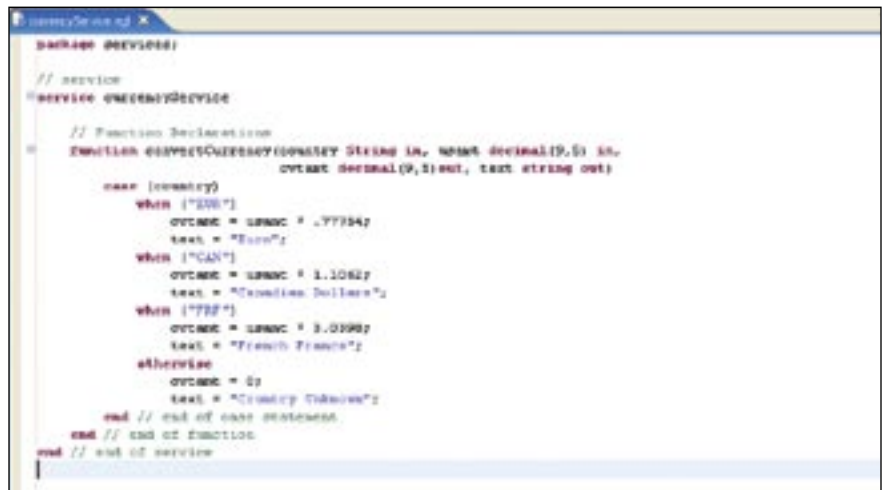
In the wizard, name the service (we called ours “currencyService”). EGL will create a template service with a template function. Just customize the template to meet your requirements.



```
package services;  
  
// service  
@service currencyService  
  
    // Variable Declarations  
    variableInString;  
  
    // Function Declarations  
    function functionInString()  
    end  
  
end
```

Figure 2: EGL service template

We will now replace the function prototype with our currency conversion function.



```
package services;  
  
// service  
@service currencyService  
  
    // Function Declarations  
    @function @convertCurrency(@country String in, @rate decimal(9,5) in,  
                                @output decimal(9,5)out, text string out)  
  
    case [country]  
        when ("USD")  
            output = @rate * .7754;  
            text = "Euro";  
        when ("CAN")  
            output = @rate * 1.1052;  
            text = "Canadian Dollars";  
        when ("FRF")  
            output = @rate * 3.0339;  
            text = "French Francs";  
        otherwise  
            output = 0;  
            text = "Country Unknown";  
    end // end of case statement  
end // end of function  
end // end of service
```

Figure 3: Completed service

Our goal is to demonstrate the simplicity of creating a service. The key point of this example is that we can determine our service input parameters, and the system will select the appropriate calculation and return two output parameters to the invoking program.

Depicted in figure 3 is our `currencyService`. This service has a single function called “`convertCurrency(*)`.”

It will have two input parameters:

1. *country* – a string that will contain the code for the country to whose currency we wish to convert
2. *usamt* – the amount in U.S. dollars that we wish to convert

There are two output parameters returned from the service:

1. *cvtamt* – the converted amount based on the exchange rate for the country
2. *text* – a character string that describes the result

The code uses a simple CASE expression to determine the currency conversion to use. The code then performs a simple calculation and returns two parameters as the result.

While this example is deliberately simplistic for illustration purposes, EGL allows you to build sophisticated applications using many functions, input/output (I/O) events, function Libraries and calls to other programs that may exist in your environment, all in the context of an EGL service.

You may now define the deployment information necessary to deploy your service and proceed to test your service.

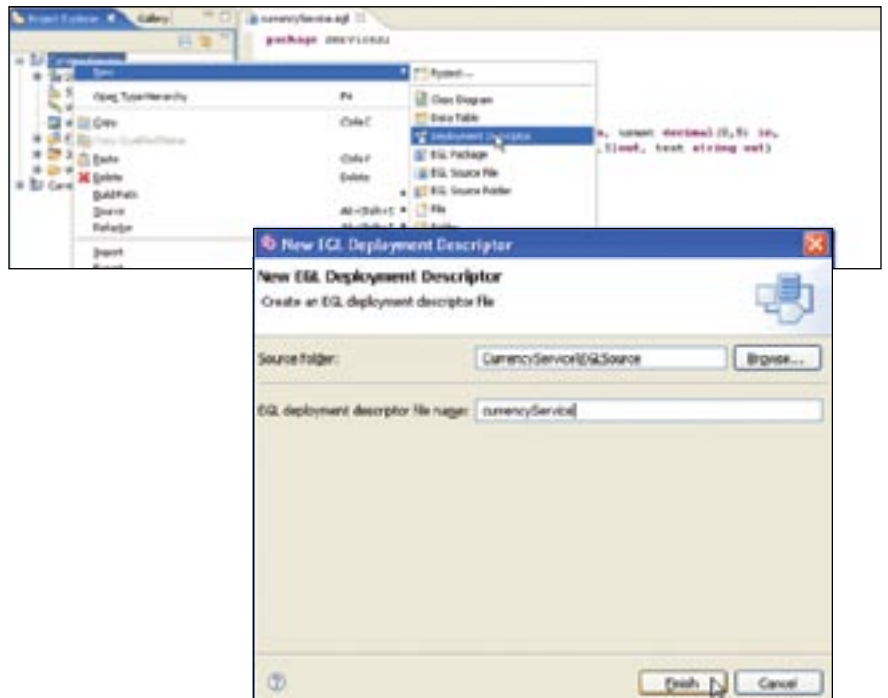


Figure 4: Create a Deployment Descriptor.

To create the Deployment Descriptor for your service:

1. Select your project (“CurrencyService”) in the Navigator display.
2. Right-click.
3. Select *New*.
4. Select *Deployment Descriptor*.
5. Name the Deployment Descriptor (“currencyService”).
6. Click *Finish*.

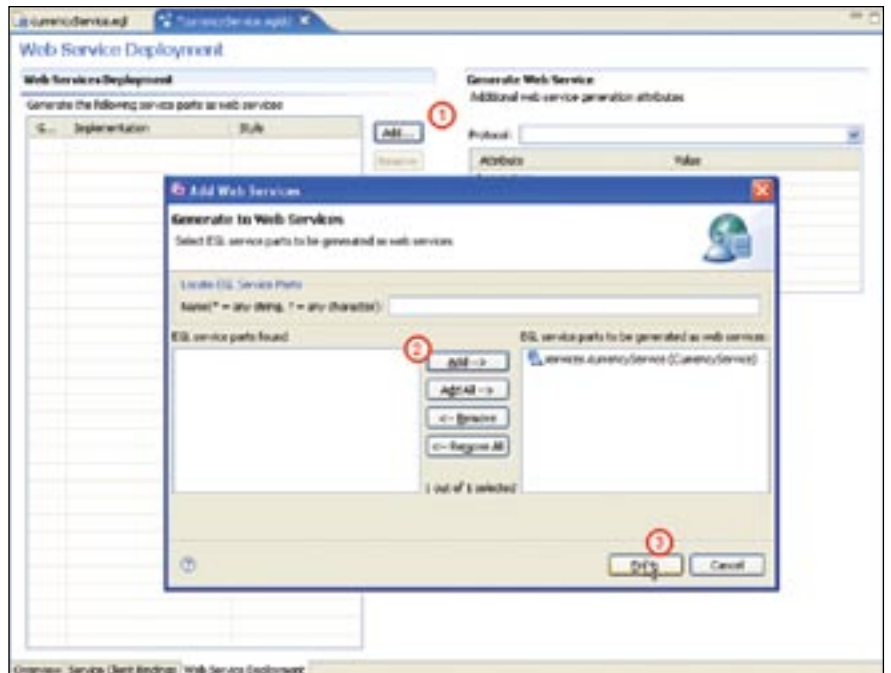


Figure 5: Add a service to the Deployment Descriptor.

From the Deployment Descriptor view, be certain to select the Web Service Deployment tab. Then:

1. Click *Add* (item 1 in figure 5).
2. Select the service from the window on the left of the dialog (item 2 in figure 5), and click *Add*.
3. Click *Finish*.
4. Save the Deployment Descriptor.

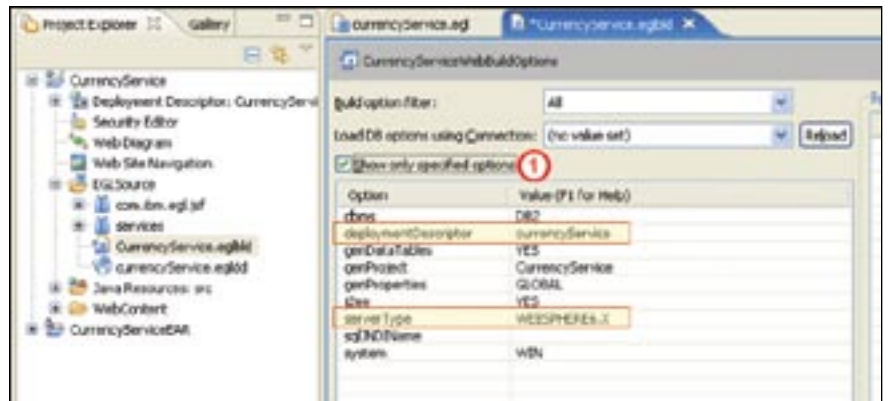


Figure 6: EGL Build Descriptor

We must now link the Deployment Descriptor to the EGL Build Descriptor.

1. Locate the EGL Build Descriptor in your EGL source folder (“CurrencyService.eglbl”). Double-click to open the Build Descriptor.
2. Uncheck the *Show only specified options* checkbox to see all options.
3. Locate *deploymentDescriptor*. Click the down arrow and select *currencyService* from the list.
4. Locate *serverType*. Choose the server type on which you’re going to deploy or test your service. We have chosen “WEBSPHERE6.X.”
5. Click the *Show only specified options* checkbox, and your display should look similar to the one shown in figure 6.
6. Save the EGL Build Descriptor.

Now we are ready to generate and test our Web service.

1. Select *CurrencyService* in the Project Explorer view.
2. Right-click and select *Generate*.
3. A dialog will appear. Messages will indicate that your service has been deployed.

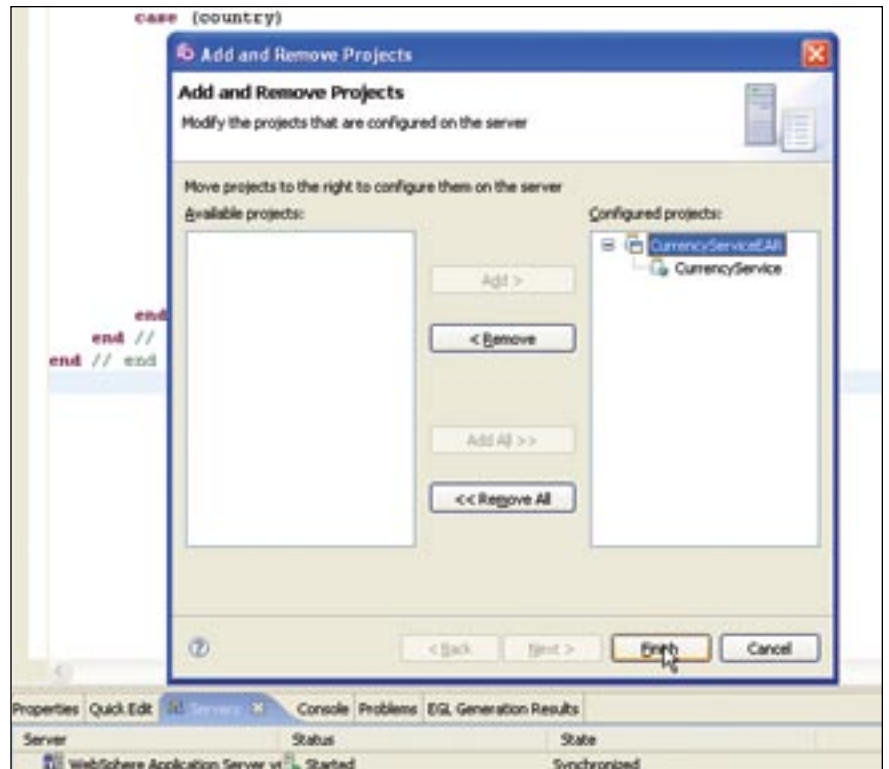


Figure 7: Start the IBM WebSphere Test Environment.

To test your service, you will need to select the IBM WebSphere Application Server option, right-click and start it.

1. Right-click the server; then select *Add and Remove Projects*.
2. Select *CurrencyServiceEAR* and click *Add*.
3. Click *Finish*.

To test your Web service, use the Web Services Explorer tool.

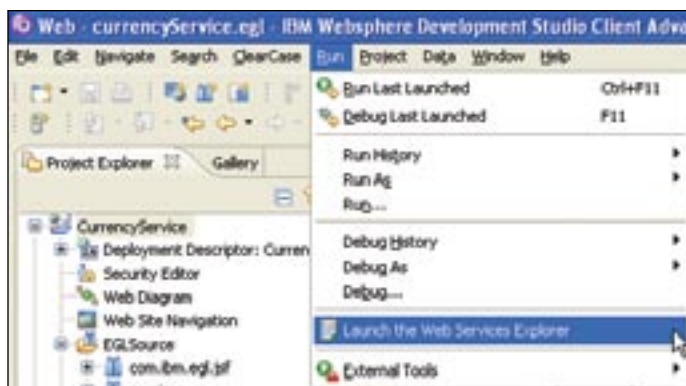


Figure 8: Launch Web Services Explorer.

Select and click *Launch Web Services Explorer* from the Run menu in your workspace.

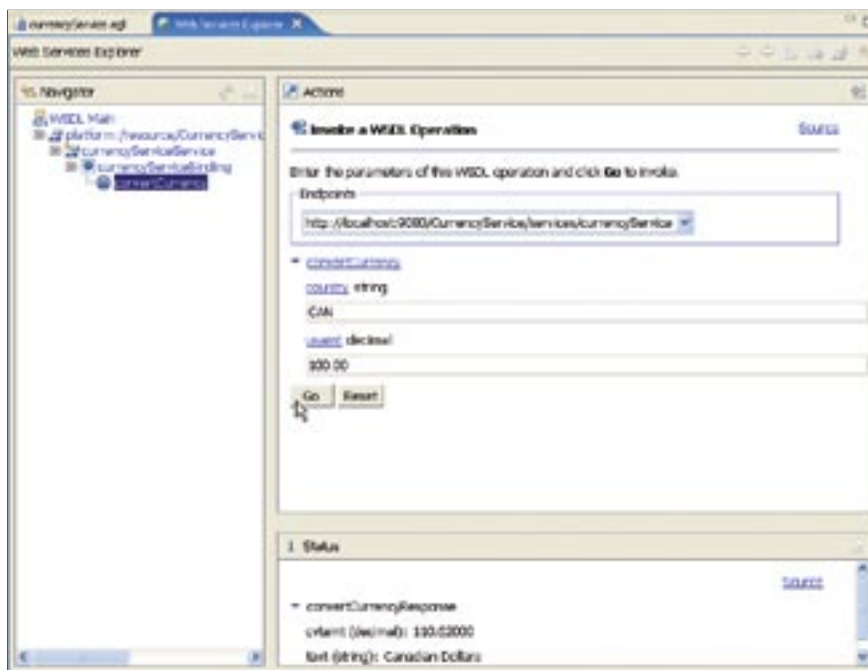


Figure 9: The Web Services Explorer tool

To utilize the Web Services Explorer tool, click *WSDL* highlighted in the upper right corner of figure 9.

1. Click *Browse*.
2. Select *CurrencyService*.
3. Click the *convertCurrency* method link in the display to see your input parameters.
4. Type in a country code (recall that we specified “EUR,” “CAN” and “FRF” when we created our service; see figure 3).
5. Type in a dollar amount in U.S. dollars.
6. Click *Go*. The results will be displayed in the Status window at the bottom of the screen.
7. Be sure to test all of the options, including an invalid country code or amount.

Congratulations, you have created and tested a Web service! You may now deploy and install your EAR file on the server of your choice.

One note about services: By definition, standard and convention, services cannot store variables between executions. This means that every time you execute a service, it's the first time. Services have no concept of transaction or persistence. EGL will initialize all variables in the service upon completion of the execution of the service.

Consuming a service in EGL

Now that we have created a functional service, we will consume the service in a JavaServer Faces (JSF) Web page.

To consume the service we just created in a JSF technology-based Web page:

1. Open the project in your workspace containing the service we previously created.
2. Create a new EGL Web project for the Web application that will use the service.
3. Create a Deployment Descriptor using the Service Client Bindings.
4. Update the EGL Build Descriptor to point to the Deployment Descriptor.
5. Create your Web page using tools to drag and drop your service onto the page.
6. Save and test your Web application.

Create a new Deployment Descriptor

Now we'll go through the steps to create a new Deployment Descriptor.

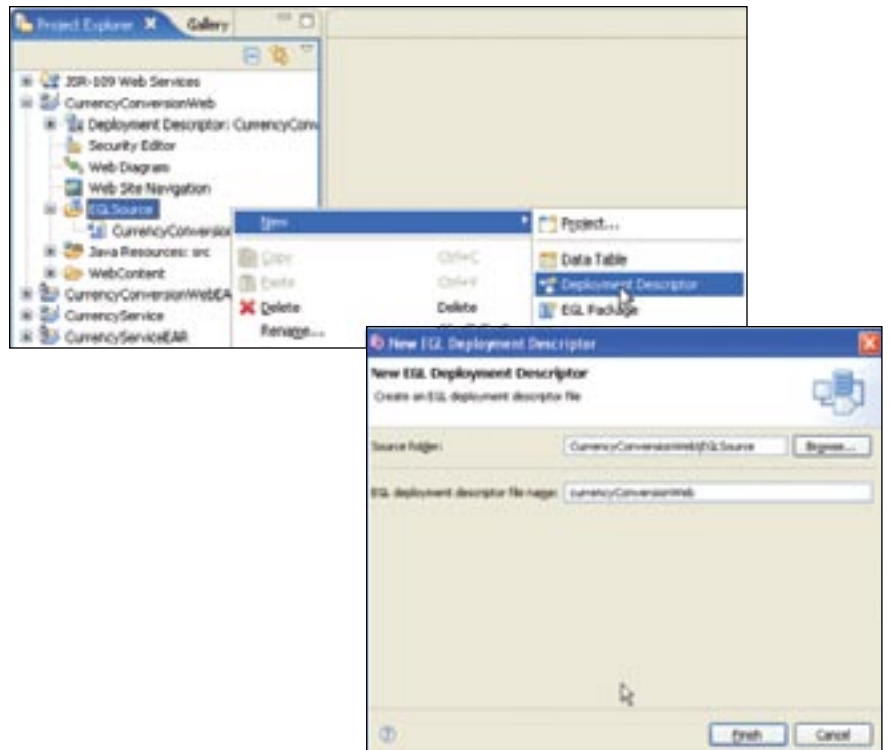


Figure 10: Create a Deployment Descriptor.

1. Right-click on your project or EGL source.
2. Select *New*.
3. Select *Deployment Descriptor*.
4. In the dialog, name your Deployment Descriptor. We named ours “currencyConversionWeb.”

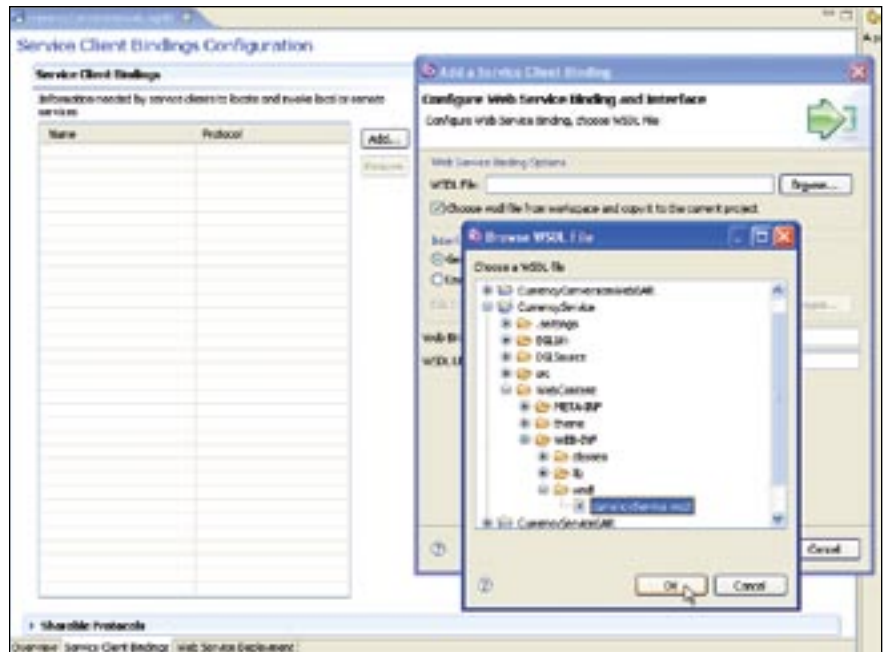


Figure 11: Point to WSDL.

5. Select the Service Client Bindings tab on the Deployment Descriptor view.
6. Click *Add*.
7. Click the *Choose wsdl file from workspace and copy it to the current project* checkbox.
8. Click *Browse*.
9. Locate your WSDL file in the CurrencyService project as illustrated in figure 11, and click *OK*.
10. Click *Finish*. The Deployment Descriptor will be created and the WSDL file will be copied into your project.

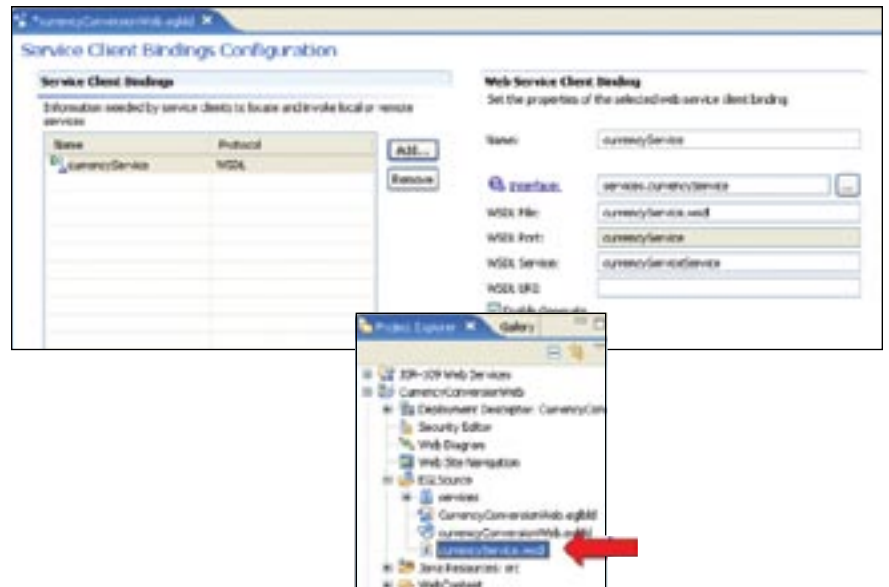


Figure 12: Results of Deployment Descriptor wizard

After completing the wizard, your Deployment Descriptor should look like the upper illustration in figure 12.

Note that the WSDL file was copied into the EGL source folder in the CurrencyConversionWeb project.

Update the EGL Build Descriptor
Now we'll update the Build Descriptor.

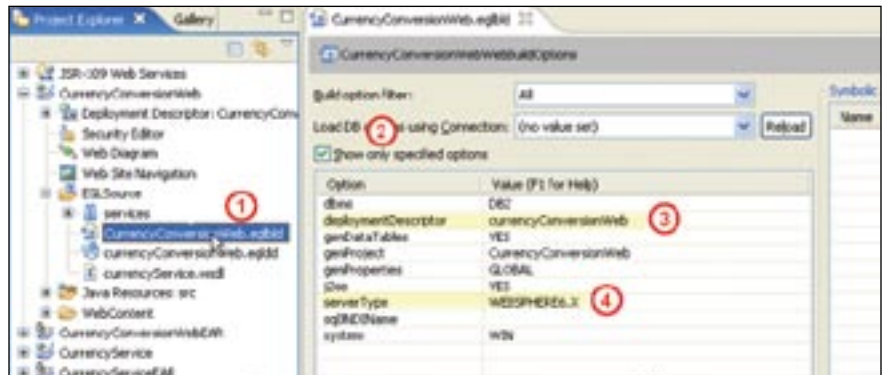


Figure 13: EGL Build Descriptor

1. Open the EGL Build Descriptor by double-clicking on its name in the Project Explorer tool (“CurrencyConversionWeb.eglbld”).
2. Uncheck the *Show only specified options* checkbox.
3. Locate the *deploymentDescriptor* option and use *Browse* in the entry field to select the name of your Deployment Descriptor.
4. Locate the *serverType* option. Click in the entry field; then click the drop-down list button and select the application server you are using. We chose “WEBSPPHERE6.X.”
5. Save and close the EGL Build Descriptor.

Create an EGL Web page
Now we'll create an EGL Web page.

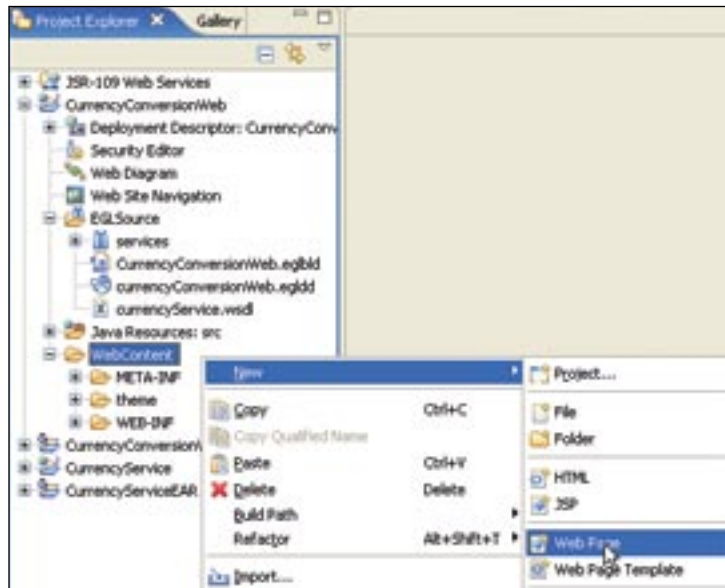


Figure 14: Create an EGL Web page.

1. Select *Web Content* in your Web project.
2. Right-click and select *New*.
3. Click on *Web Page*.

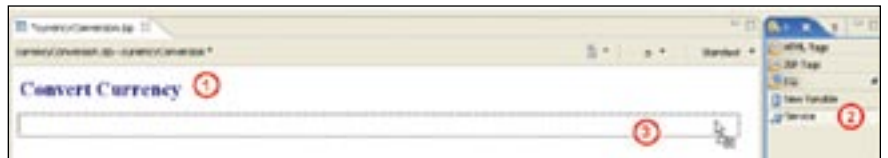


Figure 15: EGL Web page

4. In the Web designer, type a title for your page. We used “Currency Conversion” and edited it in the Properties tab at the bottom of the workspace.
5. Click *Services* in the tools palette.
6. Click in the rectangular box at the circled number 3 in figure 15. This will cause the service to appear in the Page Data view at the lower left corner of your workspace.

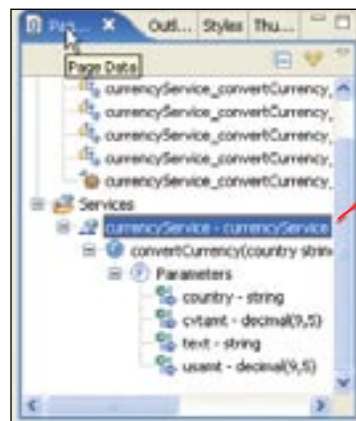
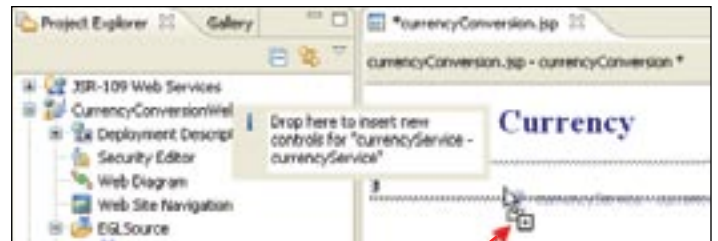


Figure 16: Drag a service to the Web page.

7. Select your service in the Page Data view in your workspace.
8. Drag and drop the service in the rectangle on the Web page.

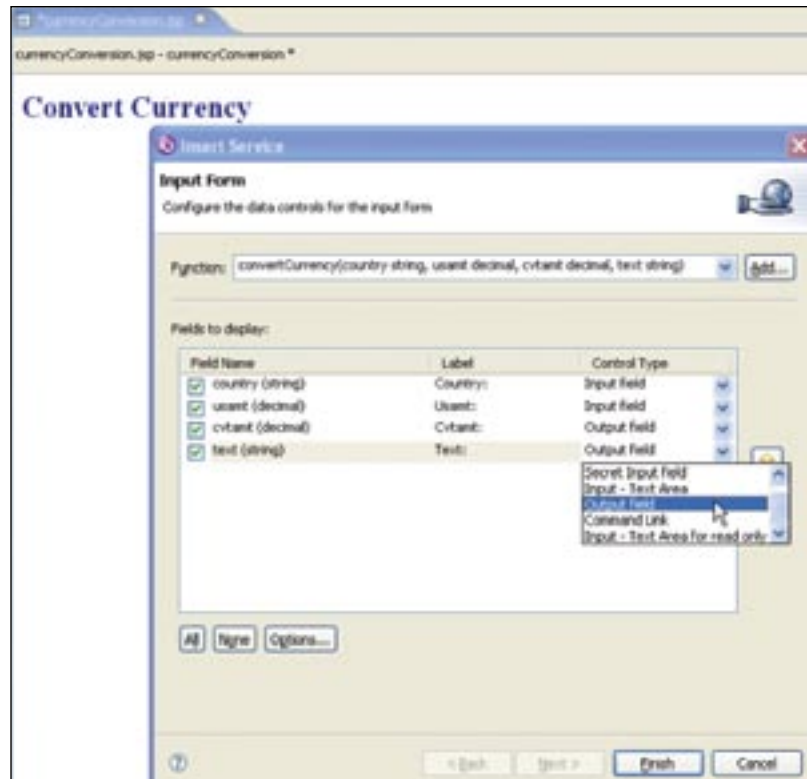


Figure 17: Adjust field types.

9. Change the field type for *cvtamt* from an input field to an output field.
10. Change the field type for *text* from an input field to an output field.
11. Click *Finish*.

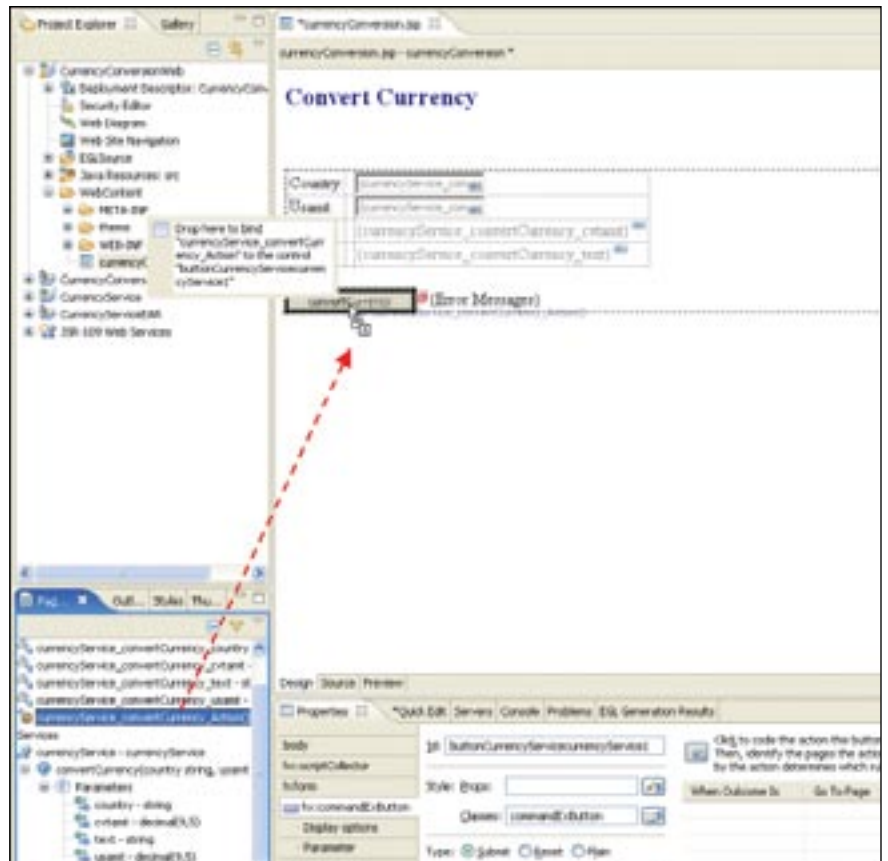


Figure 18: Drag and drop action on button.

12. Select `currencyService_convertCurrency_Action()` in the Page Data view.
13. Drag and drop the action onto `Submit` in the Web page.

You have now completed the steps to create a Web page consuming a Web service that you had previously created. You may now test your application by running the application on the WebSphere test server.



```
package jpfhandlers;
import services.*;

@handler currencyConversion type JPFHandler
@contextHandlerFunction = @contextHandler,
view = "currencyConversion.xpt"

currencyService_convertCurrency_country string;
currencyService_convertCurrency_currency decimal(9, 5);
currencyService_convertCurrency_rate decimal(9, 5);
currencyService_convertCurrency_text string;
currencyService_currencyService (@contextHandler);
// Function Declarations
function @contextHandler();
end

function currencyService_convertCurrency_action()
currencyService_convertCurrency(currencyService_convertCurrency_country, currencyService_convertCurrency_currency, currencyService_convertCurrency_rate, currencyService_convertCurrency_text);
end
end
```

Figure 19: Generated EGL code

Right-click anywhere in the Web page and select *Edit Page Code* if you wish to display or customize the EGL page handler code generated by the tools. Figure 19 illustrates the unmodified code that has been generated by the IBM Rational Business Developer tool for you.

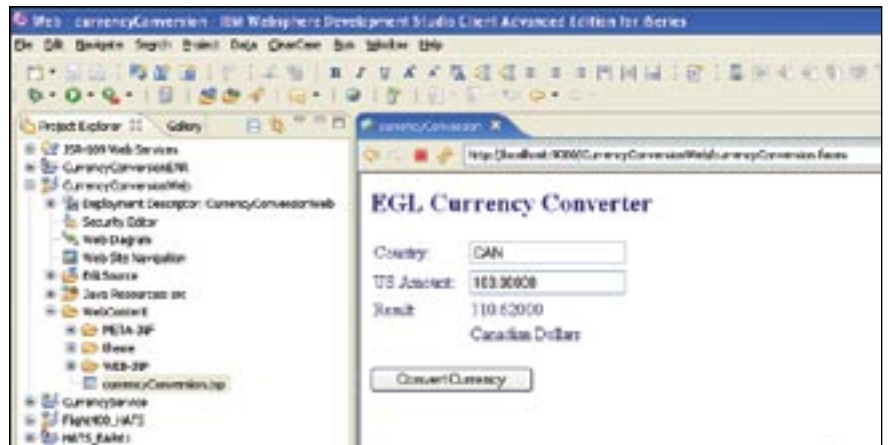


Figure 20: Running application in browser

Figure 20 illustrates the result of our example. We entered “CAN” as the target country, 100.00 U.S. dollars and clicked *Convert Currency*. As you can see, it converted the amount to 110.62 Canadian dollars.

EGL is committed to SOA

SOA is a generic conceptual architecture. While it uses standards from many disciplines, there are no SOA standards that define it. IBM, BEA, IONA, Oracle, SAP, Siebel Systems and Sybase are leveraging Eclipse technology and a set of emerging standards to create Service Component Architecture, or SCA. Unlike SOA, SCA will rely on a more robust framework and standards—rather than Web services—to help ensure the interoperability of companies that wish to adopt SOA.

For more information

To see an introduction to SCA, visit:

ibm.com/developerworks/library/specification/ws-sca

To view information on the work that Eclipse is performing, visit:

www.eclipse.org/stp

You can count on EGL to provide levels of abstraction and ease of use similar to those you have seen in this paper. EGL will continue to hide technical complexity, allowing customers to focus on the business problem at hand.

This paper is an introduction to EGL's implementation of SOA. EGL has much deeper capabilities than we could describe here. For more information on Rational EGL functionality and IBM Rational Business Developer Extension software, visit:

ibm.com/software/awdtools/developer/business/index

Or visit:

ibm.com/developerworks/rational/products/rbde



© Copyright IBM Corporation 2007

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
06-07
All Rights Reserved.

AIX, i5/OS, IBM, the IBM logo, Rational, System i, System z and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product and service names may be trademarks or registered trademarks or service marks of others.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

This publication contains other company Internet addresses. IBM is not responsible for information found on these Web sites.