Create and consume Web services using a simple and powerful tool
June 2006

SOA made simple with EGL.

Bob Cancilla
Product Market Manager, Rational Tools for System i and System z

Contents

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. 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, EGL 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.

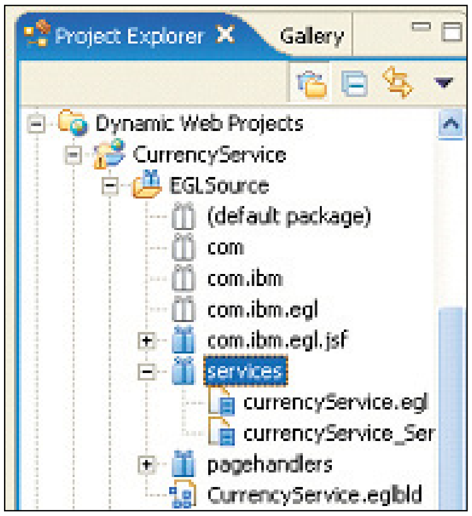
Following is a simple example of creating a Web service with EGL, and then consuming that service.

Creating 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 IBM WebSphere Development Studio Client for iSeries™ Advanced Edition software.

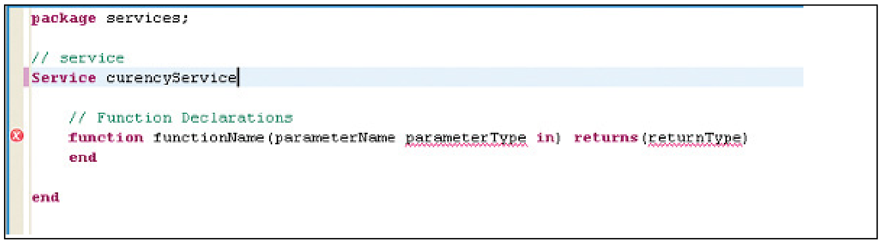
To create a service in EGL:

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 Service Explorer.*



First, create an EGL dynamic Web project. We named our project "Currency-Service." 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."

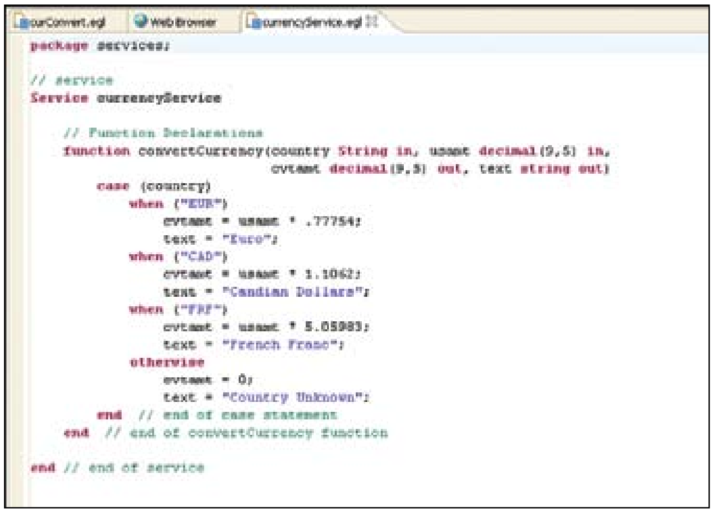
In the wizard, name the service (we called ours "currencyService"). EGL will create a template service with a template function. Just customize the template to meet your requirements.



```
package services;

// service
Service curencyService

    // Function Declarations
    function functionName(parameterName parameterType in) returns(returnType)
    end

end
```

Figure 1: 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, usamt decimal(9,5) in,
                             cvtamt decimal(9,5) out, text string out)
        case (country)
            when ("EUR")
                cvtamt = usamt * .77754;
                text = "Euro";
            when ("CAD")
                cvtamt = usamt * 1.1062;
                text = "Candian Dollars";
            when ("FRF")
                cvtamt = usamt * 5.05983;
                text = "French Franc";
            otherwise
                cvtamt = 0;
                text = "Country Unknown";
        end  // end of case statement
    end  // end of convertCurrency function

end // end of service
```

Figure 2: Completed function

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 2 is our currencyService. This service has a single function called "convertCurrency(*)." It will have two input parameters:

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

There are two output parameters returned from the service:

- *cvtamt—the converted amount based on the exchange rate for the country*
- *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.

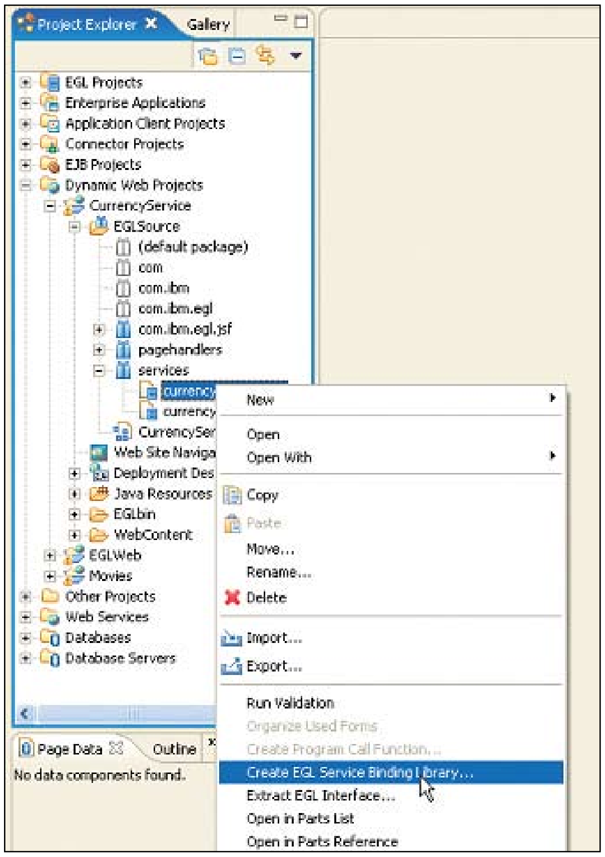
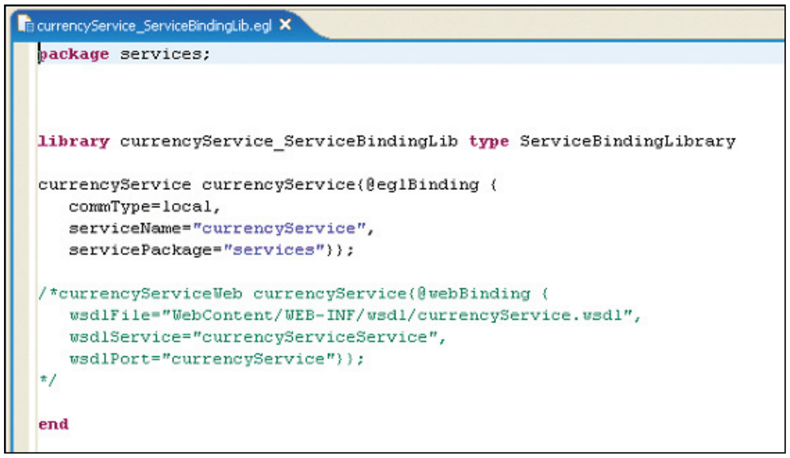


Figure 3: Generate the EGL Service Binding Library.

The next step is to save the service created. Select the Project Explorer view, right-click in the view and select “Create EGL Service Binding Library...” from the context menu.



```
package services;


library currencyService_ServiceBindingLib type ServiceBindingLibrary

currencyService currencyService(@eglBinding {
   commType=local,
   serviceName="currencyService",
   servicePackage="services"});

/*currencyServiceWeb currencyService(@webBinding {
   wsdlFile="WebContent/WEB-INF/wsdl/currencyService.wsdl",
   wsdlService="currencyServiceService",
   wsdlPort="currencyService"});
*/

end
```

Figure 4: Service Binding Library

EGL generates all of the underlying XML, WSDL and program code necessary to deploy and run your service.

You may now test the service in your development environment using the Web Service Explorer tool built into the program.
]

Consuming a service with EGL

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

We start by expanding the project we just created. Select the WebContent folder, right-click and select "New." Now select "Faces JSP File" from the menu. In the wizard that follows, we provide a name and select a page template—which will give us our look and feel.

To consume a service in EGL via a JSF Web page:

1. *Use a service in the same or different project, or import a WSDL file into the project.*
2. *Create a JSF JavaServer Pages (JSP) file.*
3. *Use the Page Data view to import a service into your JSF JSP file.*
4. *Drag and drop the service onto the Web page.*
5. *Drag and drop the function you wish to execute on top of the Submit button.*
6. *Run the Web page on your integrated WebSphere test environment server.*

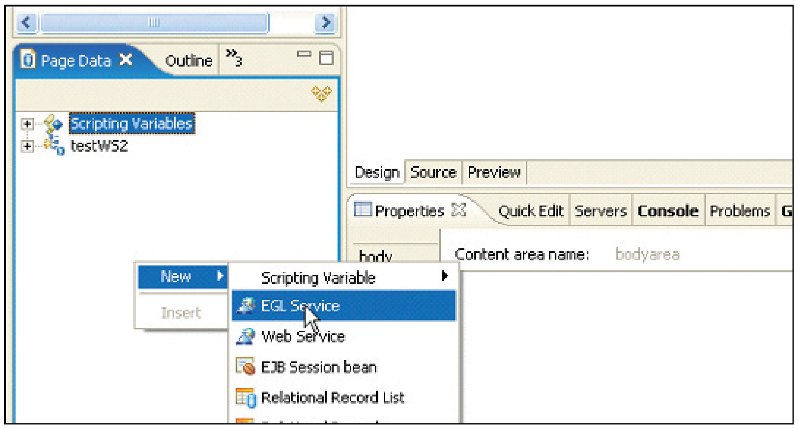
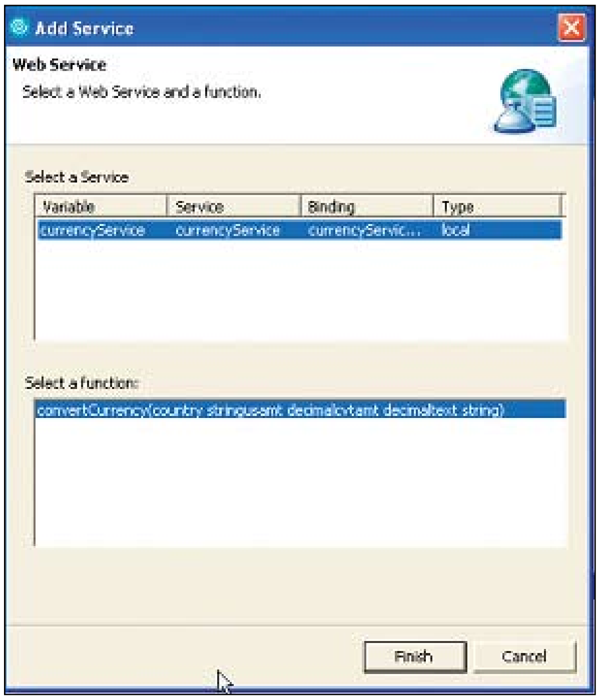


Figure 5: Import the EGL service.

In the new JSF JSP file workspace that's open on your desktop, click in the Page Data view. Now, right-click and select "EGL Service" from the menu.



Add Service

Web Service

Select a Web Service and a function.

Select a Service

Variable	Service	Binding	Type
currencyService	currencyService	currencyServic...	local

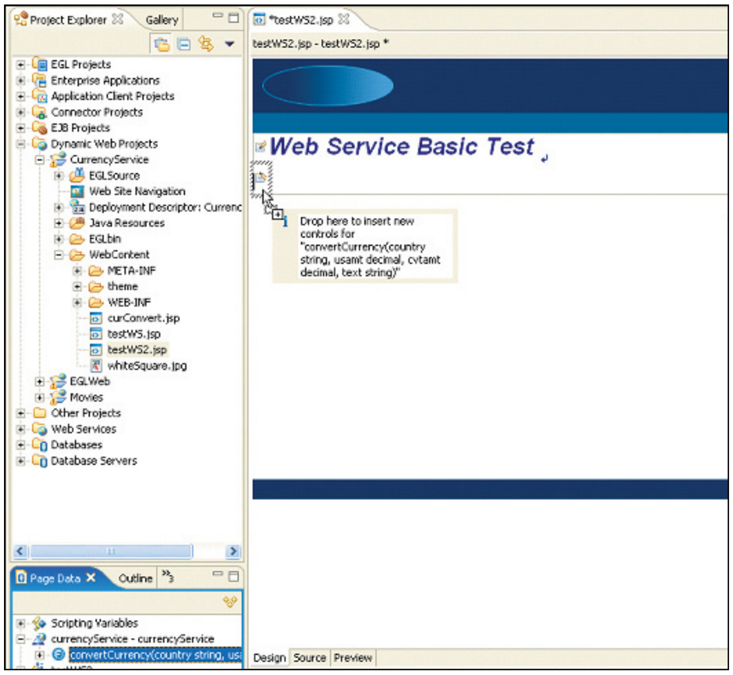
Select a function:

convertCurrency(country stringusamt decimalcvtamt decimaltext string)

Figure 6: Add the service.

The Add Service dialog appears. In this case, there is the single service that we just created and the single function "convertCurrency(*)." If there were other services available to the project or other functions within the service, they would be displayed and available for selection. In our case, we do nothing but click the Finish button.

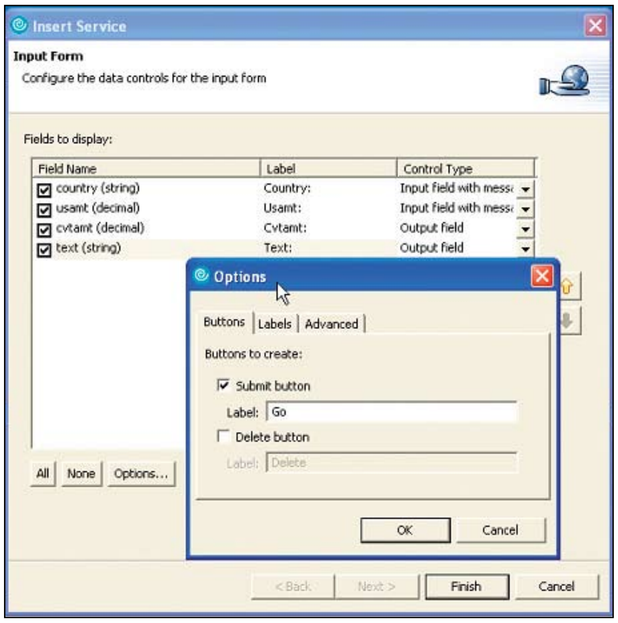
You will now see the service that we created appear in the Page Data view. We need only drag and drop the service onto the JSF Web page in the large Page Designer view of the tools.



Drop here to insert new controls for "convertCurrency(country string, usamt decimal, cvtamt decimal, text string)"

Figure 7: Drag and drop the service on the Web page.

As you can see in figure 7, we select the service in the Page Data view (lower left corner of the figure; note the blue circled “F”). We drag and drop the service onto the Web page as illustrated.


Insert Service

Input Form
Configure the data controls for the input form

Fields to display:

Field Name	Label	Control Type
☑ country (string)	Country:	Input field with messa
☑ usamt (decimal)	Usamt:	Input field with messa
☑ cvtamt (decimal)	Cvtamt:	Output field
☑ text (string)	Text:	Output field

Options

Buttons to create:

☑ Submit button
Label: Go
☐ Delete button
Label: Delete

Figure 8: The Insert Service dialog

As illustrated in figure 8 above, EGL will discover and list the input and output parameters. Because of the nature of the XML or WSDL file, EGL will display all of the parameters as input. Since we know that "cvtamt" and "text" are output parameters, change them to output fields as indicated in the illustration. Note that the illustration also shows that we have clicked the Options button and changed the name of the Submit button (to "Go") generated on the page.

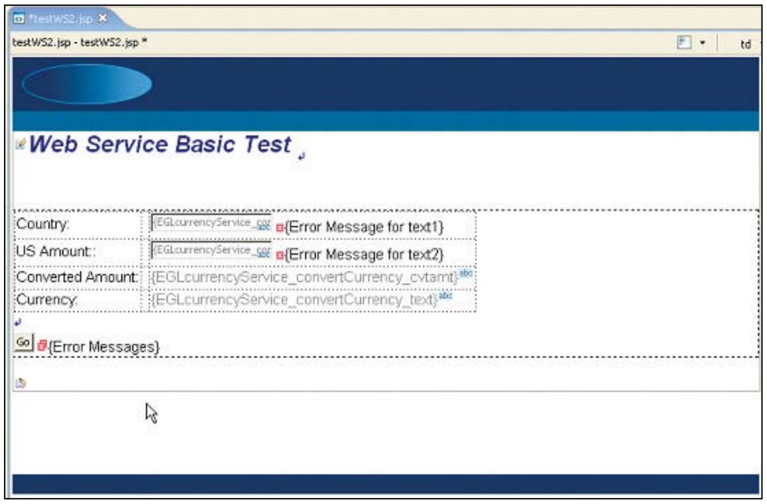

*testWS2.jsp
testWS2.jsp - testWS2.jsp *
Web Service Basic Test
Country:
US Amount:
Converted Amount:
Currency:
{EGLcurrencyService_convertCurrency_cvtamt}
{EGLcurrencyService_convertCurrency_text}
{Error Message for text1}
{Error Message for text2}
Go
{Error Messages}

Figure 9: Generated Web page

Figure 9 illustrates the Web page with the fields and data bindings generated from the service by the wizard.

The last step in the process is to bind the function "convertCurrency(*)" to the Go (Submit) button.

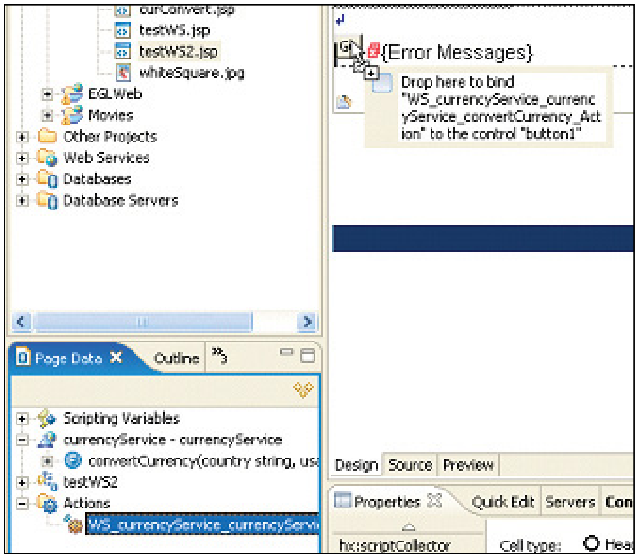


Figure 10: Bind the function to the Go (Submit) button.

Once you have bound the function to the Go button, save your JSP file and run it in the integrated WebSphere test environment.

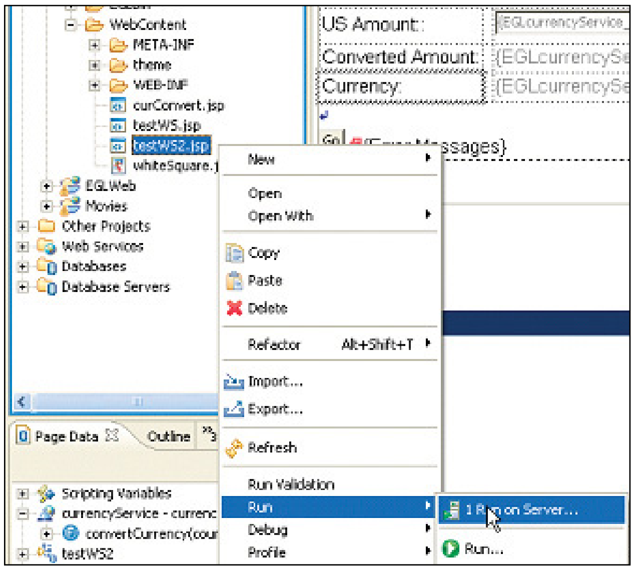
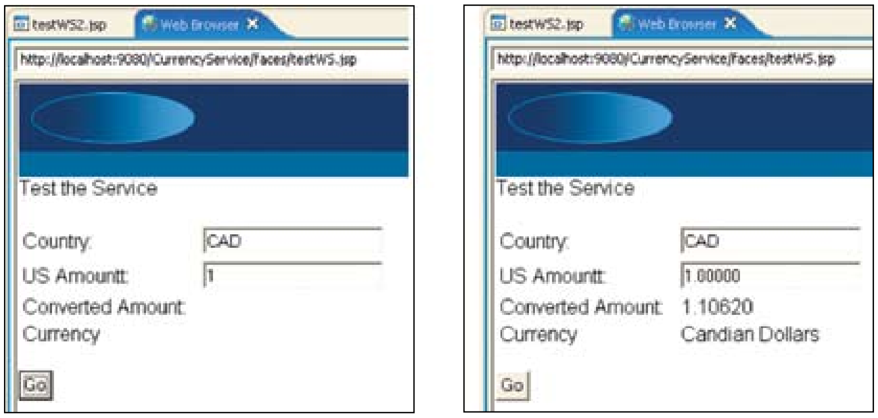


Figure 11: Run your JSP file on the server.

Select your JSP file in the WebContent folder (figure 11). Right-click on the file icon, select “Run” and then select “Run on Server…” If this is the first time you are running the file, a dialog will appear. Click “Finish” in the dialog box.



```
http://localhost:9080/CurrencyService/faces/testWS.jsp

Test the Service

Country:            CAD
US Amountt:         1
Converted Amount:
Currency
```

```
http://localhost:9080/CurrencyService/faces/testWS.jsp

Test the Service

Country:            CAD
US Amountt:         1.00000
Converted Amount:   1.10620
Currency            Candian Dollars
```

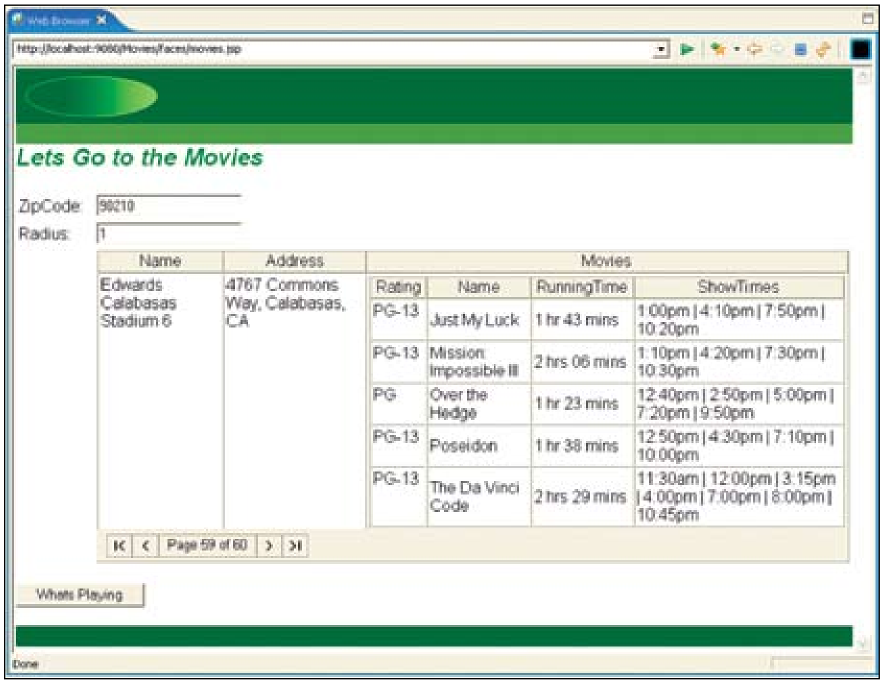
Figure 12: Finished Web page

As you can see, with only a few clicks and a bit of dragging and dropping, you have created a Web page that consumes a Web service.

EGL does more …

If you wish to consume a public Web service, first locate the WSDL file for the public Web service and save it to your workstation. Create a folder under the WebContent directory (we called ours “WSDL”) and import the external service’s WSDL file into the folder. Right-click and select “Generate EGL Binding Library….”

Now when you right-click in the Page Data view and select “EGL Service,” you will see the imported service. You can drag and drop the service just as we did in the previous example with our own Web service.



http://localhost:9080/Movies/faces/movies.jsp

Lets Go to the Movies

ZipCode: 90210
Radius: 1

Name	Address	Movies			
		Rating	Name	RunningTime	ShowTimes
Edwards Calabasas Stadium 6	4767 Commons Way, Calabasas, CA	PG-13	Just My Luck	1 hr 43 mins	1:00pm | 4:10pm | 7:50pm | 10:20pm
		PG-13	Mission: Impossible III	2 hrs 06 mins	1:10pm | 4:20pm | 7:30pm | 10:30pm
		PG	Over the Hedge	1 hr 23 mins	12:40pm | 2:50pm | 5:00pm | 7:20pm | 9:50pm
		PG-13	Poseidon	1 hr 38 mins	12:50pm | 4:30pm | 7:10pm | 10:00pm
		PG-13	The Da Vinci Code	2 hrs 29 mins	11:30am | 12:00pm | 3:15pm | 4:00pm | 7:00pm | 8:00pm | 10:45pm

Figure 13: An external Web service

Figure 13 illustrates how EGL can consume a public Web service. This service may be located at: http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.asmx?WSDL.

Notice that with this service, we input a U.S. ZIP code and radius in miles. The program returns a complex data structure consisting of two nested arrays. EGL resolves the data structures and maps them to the Web page for you.

To use the public Web service described above:

1. *Create an EGL Web project.*
2. *Create a directory for the WSDL file in the WebContent folder.*
3. *Import the WSDL file you previously downloaded into the WSDL directory.*
4. *Right-click the WSDL file and run the "Create EGL Binding Library…" wizard.*
5. *Create a JSF JSP file.*
6. *In the Page Data view, use "EGL Service" to import the service into your Web page.*
7. *Drag and drop the service onto the page.*
8. *Drag and drop the appropriate function from "Actions" in the Page Data view onto your Submit button.*
9. *Save and run the application.*

To achieve the look and feel illustrated in figure 13, you may want to edit the JSP file utilizing a number of JSF facilities.

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 ensure the interoperability of companies that wish to adopt SOA.

See an introduction to SCA at:

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

Information on the work that Eclipse is performing can be viewed at:

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, visit:

ibm.com/software/awdtools/eglcobol

G507-1948-00