



Tutorial: Create a hello world service with EGL

Version 8.5



Tutorial: Create a hello world service with EGL

Version 8.5

Note

Before using this information and the product it supports, read the information in “Notices,” on page 23.

This edition applies to version 8.5 of Rational Business Developer and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2000, 2012.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Create a hello world service with EGL. . . 1

Introduction 1

Lesson 1: Create an EGL web project for the service . 3

 Create the interface to define the service 4

 Create the service. 6

Lesson 2: Set up the service 8

 Verify the deployment descriptor and build
 descriptor 8

 Generate the WSDL file 10

Lesson 3: Create an EGL service client 13

 Create the client web page 13

Lesson 4: Set up the project as a client 16

 Set up the EGL deployment descriptor file . . . 16

 Use the service in the web page 18

Summary 21

Appendix. Notices 23

Trademarks 25

Create a hello world service with EGL

In this tutorial, you will learn how to build a simple web service with EGL. Then, you will create a client that uses this service.

Learning objectives

In this tutorial, you learn to do the following tasks:

- Create and configure an EGL project
- Create a web service with EGL
- Configure an EGL project to act as a service or as a client at run time
- Create a web page controlled by EGL
- Test an application on a web application server

Time required

90 minutes

Introduction

In this tutorial, you will learn how to build a simple web service with EGL. Then, you will create a client that uses this service.

This tutorial might require some optionally installable components. To ensure that you installed the appropriate optional components, see the System requirements list.

Service-oriented architecture is a method of organizing applications in modular pieces called *services* and *clients*. The services provide logic to the clients in the form of functions, in much the same way that EGL libraries make functions available to programs. However, in service-oriented architecture, the services are *stateless*, meaning that they do not remember interactions with a particular client. In this way, each time the service is called, it is as though that service is being used for the first time. Services are also able to provide their functionality to a wide variety of applications through the WSDL standard, promoting flexibility and code reuse.

Learning objectives

In this tutorial, you learn to do the following tasks:

- Create and configure an EGL project
- Create a web service with EGL
- Configure an EGL project to act as a service or as a client at run time
- Create a web page controlled by EGL
- Test an application on a web application server

Time required

To complete this tutorial, you will need approximately 90 minutes. If you decide to explore other facets of EGL or web services while working on the tutorial, it could take longer to finish.

Skill level

Intermediate

System requirements

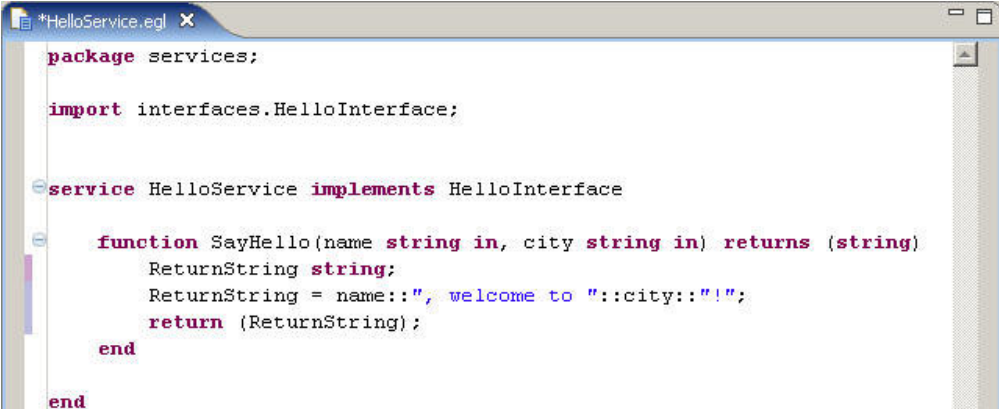
- Enterprise Generation Language
- WebSphere® Application Server

Prerequisites

There are no prerequisites for this tutorial.

Tutorial application

In this tutorial, you will create a simple web service in EGL. This service accepts a person's name and the name of a city and returns a string combining the two, such as "Bill, welcome to New York!" You will use the service-oriented architecture tools included in Rational® Business Developer Extension to expose this service as a web service and publish the information about the service in a WSDL file.



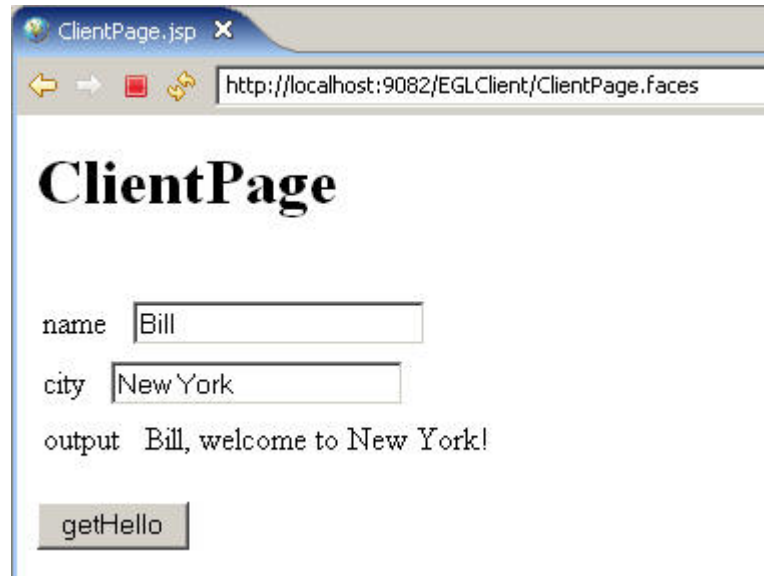
```
*HelloService.egl
package services;

import interfaces.HelloInterface;

service HelloService implements HelloInterface

    function SayHello(name string in, city string in) returns (string)
        ReturnString string;
        ReturnString = name::", welcome to ">::city::"!";
        return (ReturnString);
    end
end
```

You will then create a project to act as a client for this service. This project includes a simple web page that retrieves the two input parameters, passes them to the service, and displays the output from the service on the page:



Lesson 1: Create an EGL web project for the service

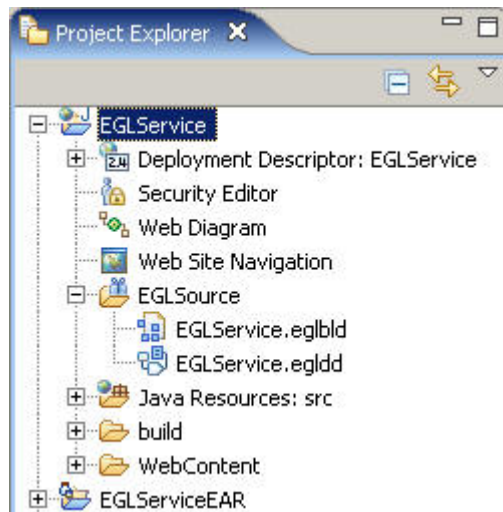
EGL projects can act as services, as clients, or as both simultaneously. For this tutorial, you will create two projects: one to act as the service and one to act as the client. While you could put all the code into a single EGL project, using two projects demonstrates how EGL can call services in another application.

Show Me

1. Optionally, you may want to use a separate workspace while working on the tutorial so you do not interfere with any of your other projects. If you want to use a different workspace, follow these optional steps:
 - a. In the workbench, click **File > Switch Workspace**. The Workspace Launcher window opens.
 - b. Enter a new workspace location in the **Workspace** field.
 - c. Click **OK**. The workbench reopens using the new workspace location. You can switch workspace locations at any time, and you can have as many workspace locations as you want.
2. Switch to the web perspective:
 - a. Click **Window > Open Perspective > Other**.
 - b. From the list of perspectives, click **Web**. If you don't see the web perspective, select the **Show all** check box.
 - c. Click **OK**.
3. Make sure EGL is set up to automatically generate deployment descriptors. Deployment descriptors contain information that describes how EGL services will be made available to other applications and how EGL applications will find services provided by other applications.
 - a. In the workbench, click **Window > Preferences... > EGL**.
 - b. Under **Default EGL Project Features Choices**, make sure that **Create an EGL deployment descriptor** is selected. If not, click this field to select it.
 - c. Click **OK**.
4. In the workbench, click **File > New > Project**.

5. In the New Project window, expand **EGL** and click **EGL Project**. If you don't see **EGL Project**, select the **Show All Wizards** check box. If you still don't see an **EGL** category or **EGL Project**, EGL is not installed on your system. Run the product setup again and select the **Additional Feature** item for EGL.
6. Click **Next**.
7. In the **Project name** field, give your project this name:
EGLService
8. Under **EGL Project Types**, click **Web Project**. This type of project allows you to use web page user interfaces.
9. Click **Next**.
10. Under **Target Runtime**, select a version of **WebSphere Application Server**.
11. Make sure that **Create a new build descriptor** is selected. Build descriptors contain options for generating your program into another language. You do not need to worry about them at this point because the wizard will create an appropriate build descriptor for you. You do not need to use the advanced settings unless you use WAS and you have previously changed the default setting that adds the project to an EAR (Enterprise Application Resource). If you use WAS, your EGLService project must belong to an EAR. The workbench remembers this setting.
12. You may see a window asking if you want to switch to the J2EE perspective. If you see this window, click **No**.

The new project appears in the Project Explorer view.



Create the interface to define the service

EGL uses the term "interface" in the same way as object-oriented languages do. For EGL, an interface defines a plan for a service to follow. Specifically, an interface contains one or more function *prototypes*, or summaries of functions. These prototypes are not usable functions themselves, but they set out plans for the real functions.

For example, suppose you needed to write an application that performed mathematical operations like a calculator. You might start by listing all of the mathematical operations your application would need (such as addition, subtraction, and multiplication), without actually writing the code to perform these operations. In this case, you would name each operation and specify each

operation's input and output parameters, but you would not start coding any logic. In this way, you would be listing the functions that the EGL service application would need in an interface. Such an interface might begin like this:

```
interface myCalculatorInterface

    //Function to add numbers together
    function addNumbers(number1 decimal(10,2) in,
        number2 decimal(10,2) in) returns (decimal(10,2));

    //Function to subtract numbers
    function subtractNumbers(number1 decimal(10,2) in,
        number2 decimal(10,2) in) returns (decimal(10,2));

end
```

Then, when you are ready to begin coding the service, you can use this interface both as a starting point and as a test to make sure you are following your plan.

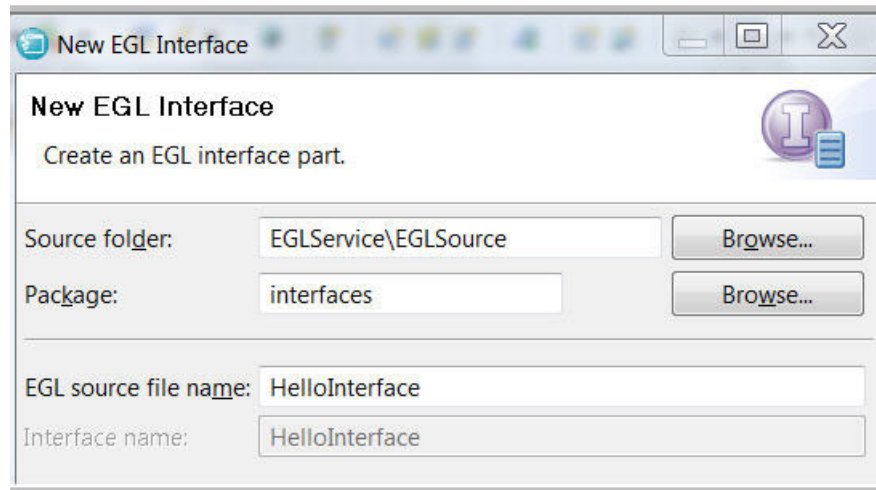
You will rarely be required to write an interface, but in general, using interfaces to describe your services is good programming practice:

- The interface lets you plan the service ahead of time, and EGL warns you if the service deviates from the interface.
- Interfaces provide a concise summary of a service, explaining what the service can do without providing all of the details of the service's implementation.
- Interfaces can serve as requirements for development or compliance.
 1. In the Project Explorer view, right-click your EGLService project to select it.
 2. Click **New > Other**.
 3. In the New window, expand **EGL** and click **Interface**. Make sure you are using the Interface item under EGL and not the Interface item under Java.
 4. Click **Next**.
 5. In the New EGL Interface window, make sure that your project's EGLSource folder is shown in the Source folder field. This field should say EGLService\EGLSource.
 6. In the **Package** field, type this name:
interfaces

EGL will create this new package because your project doesn't have a package with this name yet.

7. In the **EGL source file name** field, type this name for the new interface:
HelloInterface

The New EGL Interface window looks like this:



8. Click **Finish**. The new interface is created and opens in the EGL editor. This interface contains one function prototype already, as a sample:
`function functionName(parameterName string in) returns (int);`
9. Delete this function prototype.
10. Where the sample prototype was, insert this code for your own function prototype:
`function SayHello(name string in, city string in) returns (string);`
11. Save and close the file.

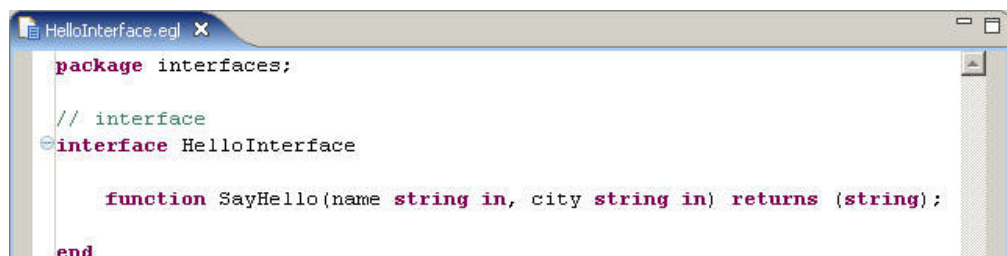
Following are some technical details about the code you just entered:

- As explained above, this is not a complete piece of EGL logic. Instead, this *function prototype* describes a function that will be in the service. In this case, the prototype describes a function named `SayHello`.
- The function accepts two pieces of input from the client, called *parameters*:
 - A string variable for a person's name
 - A string variable for a city

The function will use these parameters to put together a piece of output to return to the client.

- The code returns `(string)` indicates this return value and its type.

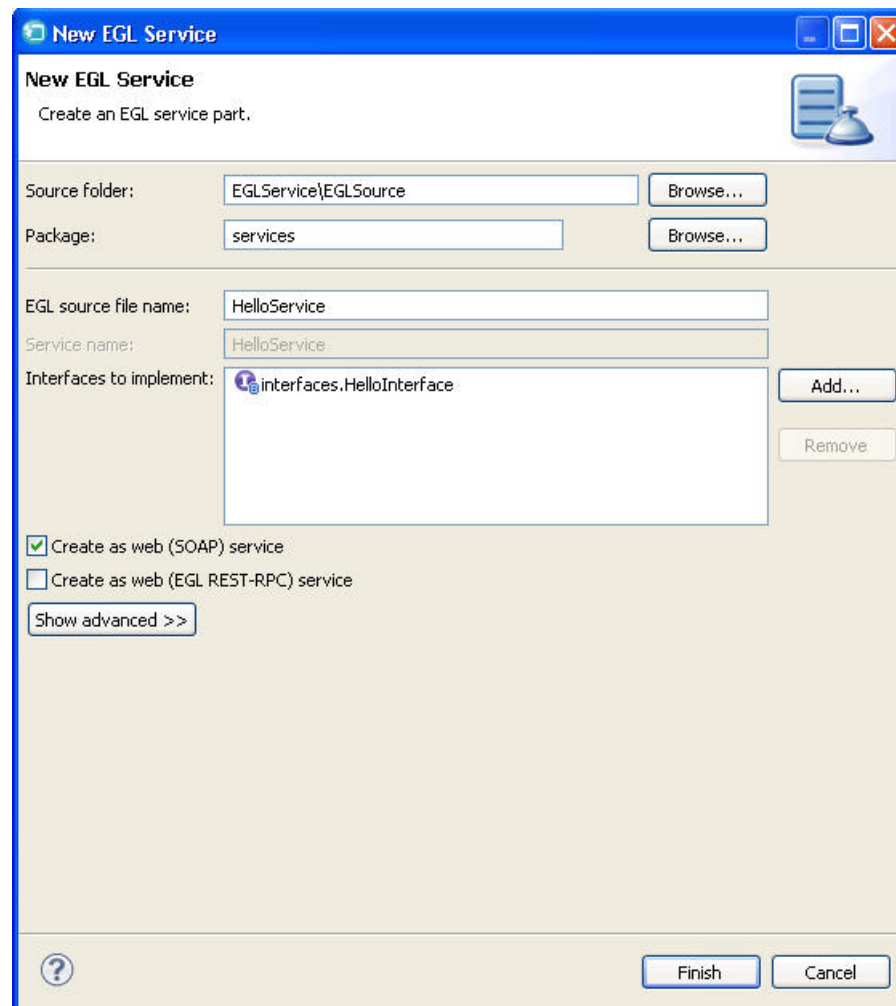
The interface looks like this:



Create the service

1. In the Project Explorer view, right-click your project and then click **New > Service**.

2. In the New EGL Service window, make sure that your project's EGLSource folder is shown in the **Source folder** field. This field should say EGLService\EGLSource.
3. In the **Package** field, type this name:
services
4. In the **EGL source file name** field, type this name for the new service:
HelloService
5. Next to **Interfaces to implement**, click **Add**.
6. Type an asterisk * in the **Choose Interfaces** field. Your interface is shown in the **Matching parts** list.
7. Click the HelloInterface from the **Matching parts** list to select it.
8. Click **OK**. The interface is now shown in the **Interfaces to implement** list.
9. Select the two check boxes to add deployment information to the deployment descriptor file. The New EGL Service window looks like this:



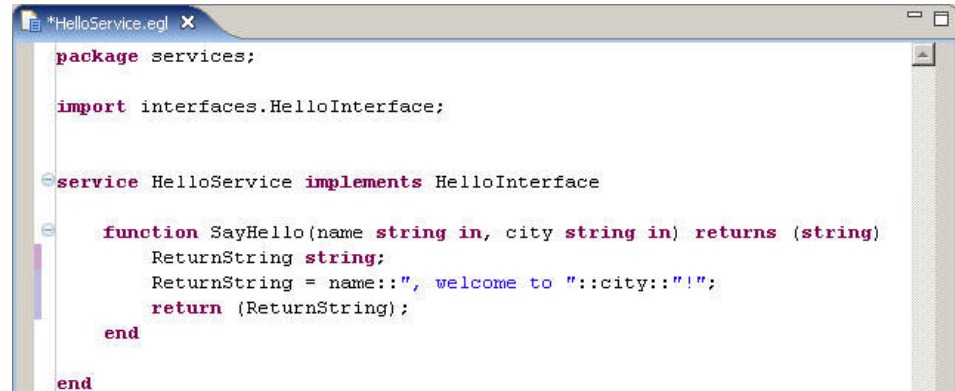
10. Click **Finish**. The new service is created and opens in the EGL editor. The service already contains a starter function based on the prototype in the interface.
11. Remove the comment `// TODO Auto-generated function`, and in its place, type the following code:

```

ReturnString string;
ReturnString = name::", welcome to "::city::"!";
return (ReturnString);

```

This code creates a string variable and assigns it a value based on the parameters, such as "Jim, welcome to Chicago!" The `::` code is a *concatenation operator*, which joins separate strings into a single string. The code looks like this:



12. Save and close the file.

Lesson 2: Set up the service

Now that you have written the code for this service, you can make it available to other applications as a web service. Making the service available in this way involves creating service binding information, which tells other applications where to find the service and what functions are available in the service. The service publishes this information as a Web Services Description Language (WSDL) file.

As an alternative to web services, EGL client applications can access EGL service applications as EGL services. This method offers better performance than web services, but it can be used only between two EGL applications. For the broadest compatibility, this tutorial uses web services, which can be used between two EGL applications, two non-EGL applications, or an EGL application and a non-EGL application. For more information, see the EGL documentation by clicking **Help > Help Contents**.

Service binding information is contained in an EGL deployment descriptor. In these steps, you work with the project's deployment descriptor and configure the project's build descriptor to use that deployment descriptor.

Show Me

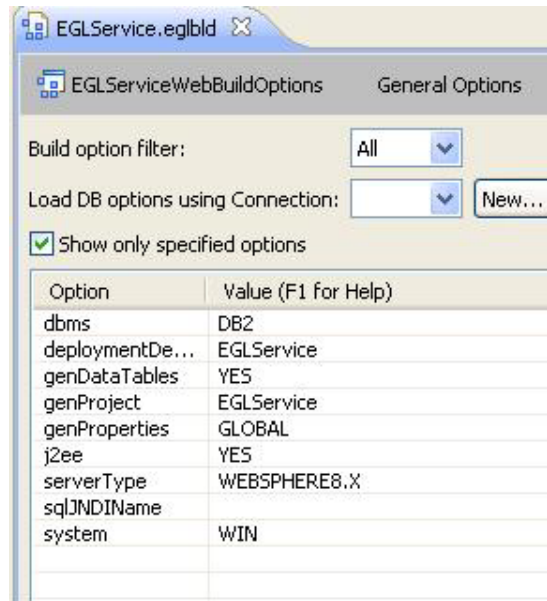
Verify the deployment descriptor and build descriptor

When you created the EGL project, you automatically created a deployment descriptor file named `EGLService.egldd`. Then, when you created the service, you selected the **Create as web service** check box, which automatically added the service to the deployment descriptor. In this section, you verify these settings.

1. In the Project Navigator view, expand the **EGLService** project and the **EGLSource** folder. Open the EGL deployment descriptor file by double-clicking the `EGLService.egldd` file.

2. In the EGL deployment descriptor editor, click the **Web Service Deployment** tab.
3. Make sure that your service is shown in the list of services to be deployed as web services, as in the following picture:

4. Close the deployment descriptor file.
5. Double-click the build descriptor for the project to open it in the build parts editor. This file is named `EGLService.eglbld` and is located in the project's `EGLSource` folder. The build descriptor file contains *build descriptor options*, which describe how EGL will generate your project into the output language.
6. In the list of build descriptor options, find the option named **deploymentDescriptor**. Note that it is set to the name of the deployment descriptor, which has the same name as the project by default. The deployment descriptor must be referenced in this way to be used. The build parts editor looks like this, with the value of **serverType** appropriate to your version of WebSphere Application Server:



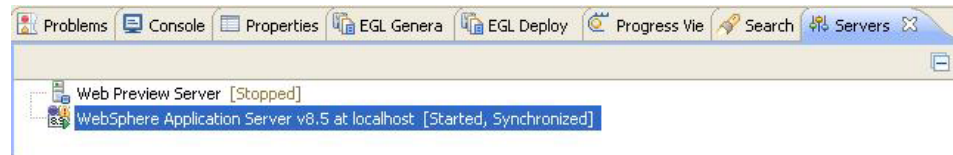
7. Close the build descriptor.
8. Generate the entire project by right-clicking the project in the Project Explorer view and then clicking **Generate**.

Generate the WSDL file

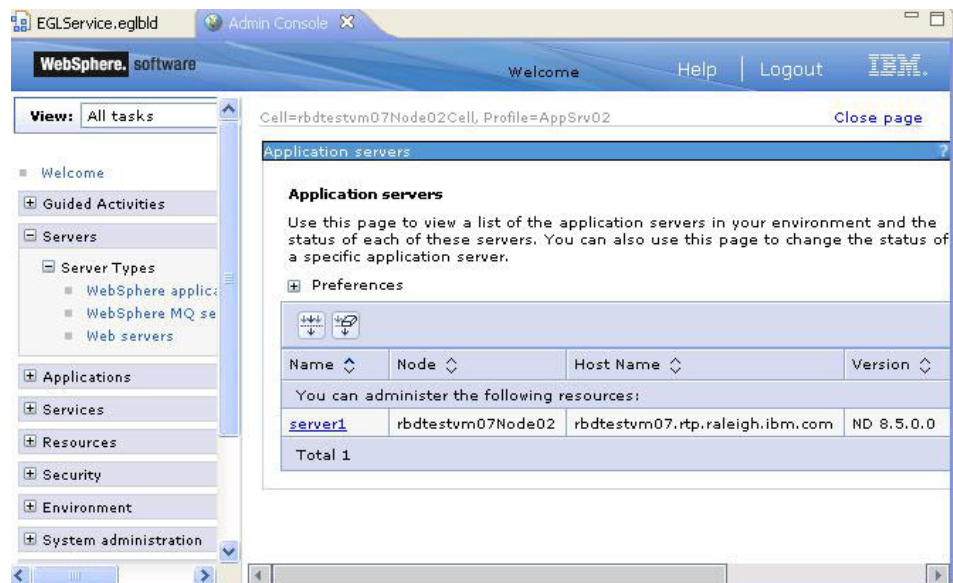
WSDL files communicate information about services to clients, describing the functions provided in the service and specifying the location of the service. In this section, you generate a WSDL file from the service. Later, your client application will import this WSDL file and use the information in it.

EGL uses the information in the deployment descriptor file and the service part itself to generate a WSDL file, but it needs one more piece of information: on which port the server will host the service. By default, the port is 9080. Follow these steps to find out the port number of your server:

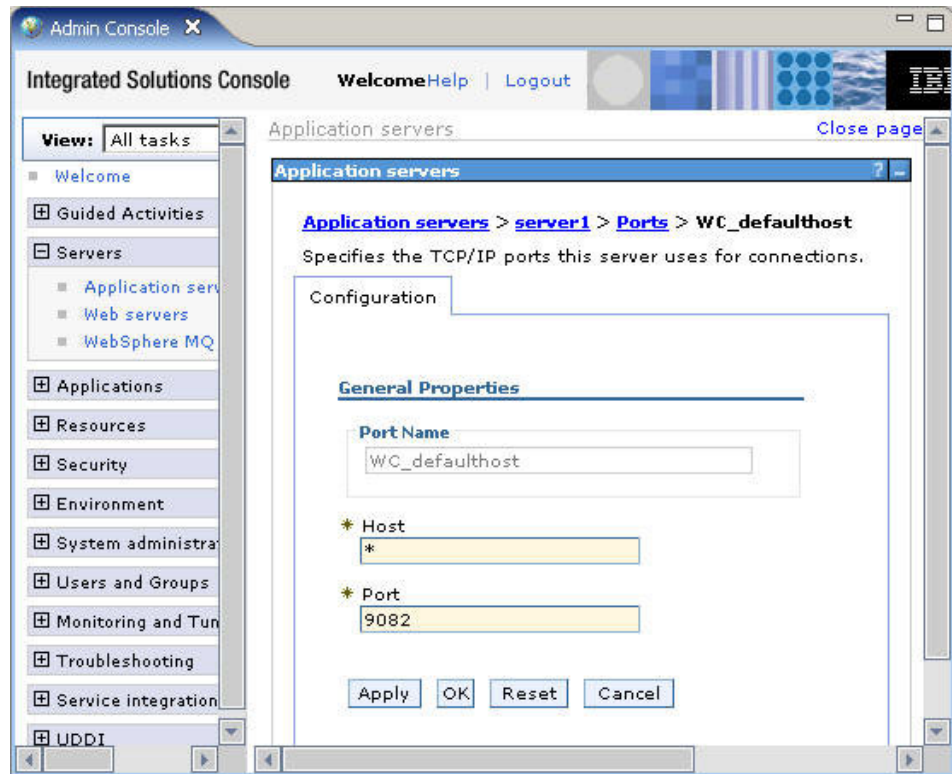
1. Open the Servers view. If you can't find the Servers view, click **Window > Show View > Servers**.
2. Right-click the server named **WebSphere Application Server**. At the popup menu, add EGLServiceEAR by clicking **Add and Remove** and then start the server by clicking **Start**. The server may take some time to start, depending on your system.
3. Wait until the server shows **Started, Synchronized** in its **Status** field, as shown in this picture:



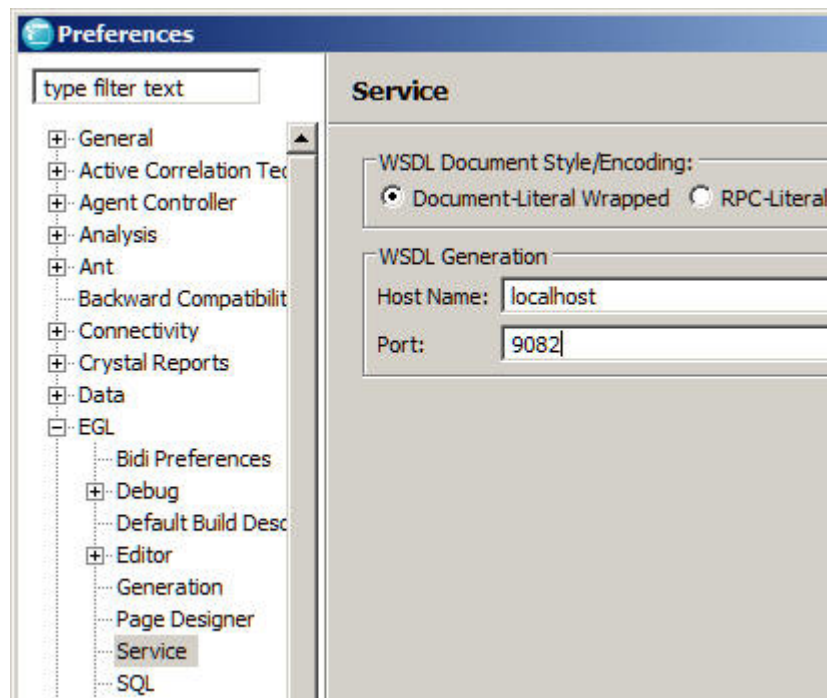
4. When the server has started, right-click it in the Servers view and then click **Run administrative console**, not Run administrative script. The administrative console opens in the editor.
5. At the left side of the administrative console, expand **Servers** and click **Application servers**. Your server is shown as a listing under **Application servers**, as in this picture:



6. Click the server name in the **Name** column.
7. On the page describing your server, click the **Configuration** tab.
8. On the Configuration tab, under **Communications**, click the **Ports** link.
9. From the list of ports, click the port with the **Port Name** labeled WC_defaulthost.
10. On the page describing this port, write down the port number in the **Port** field. In this picture, the port number is 9082.



11. Close the administrative console without making any changes.
12. In the top menu, click **Window > Preferences....** In the left panel, expand **EGL** and click **Services**.
13. If the entry for the **Port** field does not match the server port number in the administrative console, enter the port number from the console here. The window looks like this:



14. Click **Finish**.

15. In the Project Explorer view, right-click the HelloService.egl file, which is in the services package of the EGLSource folder, and then click **EGL Services > Generate WSDL File**.
16. In the Create WSDL File window, click **Finish**. EGL creates a WSDL file in the wsdl package of the EGLSource folder and displays it graphically in the WSDL editor.
17. Examine the graphical representation of the WSDL file and close it when you are finished.

Now you have configured the service to be used by other applications at run time. The WSDL file describes the service so that clients can connect to it at run time, and the deployment descriptor file allows EGL to make the service available at run time.

In the real world, services run independently of the clients that use them. To simulate this situation, you could create a new instance of the application server and run the service there. For the sake of this tutorial, there is no advantage to consuming these additional resources, so you will run the service on your existing application server at the time you test your client.

Lesson 3: Create an EGL service client

The next step is to create a client project to use the service. While the client you create in this tutorial is in the same workspace, you can imagine that it is in a completely separate location or on a different platform. Since services and clients do not need to be written in the same code language, you can also imagine that this client application is written in a different language or is created with an entirely different set of tools.

Show Me

1. Click **File > New > Project**.
2. Expand **EGL** and click **EGL Project**.
3. Click **Next**.
4. Name the new EGL project **EGLClient**.
5. Click **Web Project**.
6. Click **Next**.
7. Make sure that **Create a new build descriptor** is selected. You do not need to change the advanced settings unless you use WAS and have previously changed the settings to disable adding the project to an EAR.
8. You may see a window asking if you want to switch to the J2EE perspective. If you see this window, click **No**.

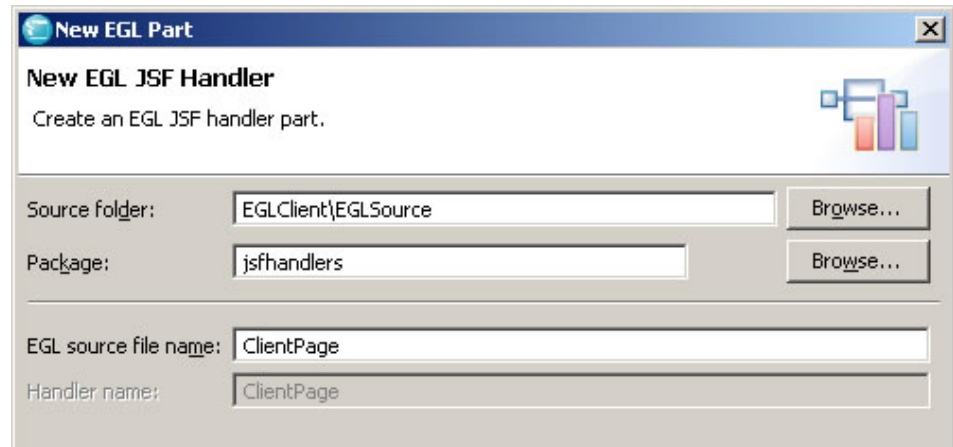
Create the client web page

To use the service, you will create a web page in the client to retrieve input, pass that input to the service, and display the output from the service. You will start by creating a JSF Handler, a type of EGL logic part that controls a web page, and then EGL will create a web page to go along with this JSF Handler.

1. In the Project Explorer view, click the EGLClient project to select it.
2. Click **File > New > Other**.
3. Expand **EGL** and click **JSF Handler**.
4. Click **Next**.

5. In the New EGL JSF Handler window, make sure that the **Source folder** field is set to the project's EGLSource folder:
EGLClient\EGLSource
6. In the **Package** field, type this name:
jsfhandlers
7. In the **EGL source file name** field, type this name:
ClientPage

The window looks like this:



8. Click **Finish**. The new JSF Handler opens in the editor.
Don't save the file until these instructions tell you to do so. When you save a JSF Handler, EGL checks to see if the web page referred to in the **view** property exists or not. If not, EGL creates the file and adds fields automatically based on the variables in the JSF Handler and the **DisplayUse** properties of those variables. If you save the file before you've added all the variables, the new web page will not include all of the variables. If this happens, you can delete the JSP file (not the JSF Handler file) and generate the JSF Handler again to get a new web page.
9. Remove the sample code from the new JSF Handler so that the following is all that is left:

```
package jsfhandlers;

handler ClientPage type JSFHandler
    {view = "jspLocation/jspName.jsp"}

end
```
10. Set the value of the **view** property to ClientPage.jsp, as in this example:

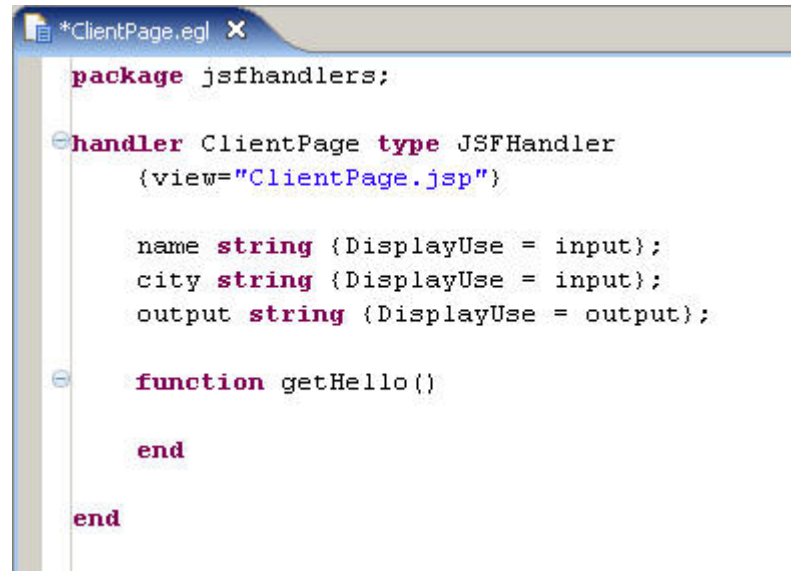
```
{view = "ClientPage.jsp"}
```
11. Within the JSF Handler, create the following variables:

```
name string {DisplayUse = input};
city string {DisplayUse = input};
output string {DisplayUse = output};
```
12. Below the variables, add this function:

```
function getHello()

end
```

You will add code to this function later. The JSF Handler looks like this:



```
package jsfhandlers;

handler ClientPage type JSFHandler
{view="ClientPage.jsp"

    name string {DisplayUse = input};
    city string {DisplayUse = input};
    output string {DisplayUse = output};

    function getHello()

end

end
```

13. Save the file. When you save the file, EGL creates a web page from the file automatically. This file is named with the value of the **view** property, ClientPage.jsp, and the file is placed in the WebContent folder of your project. If you did not get a JSP file, EGL is not configured to generate JSF Handler parts automatically. Generate the JSF Handler manually by right-clicking it either in the Project Explorer view or in the EGL editor and then clicking **Generate**. Then, follow these additional steps to set up automatic generation of JSF Handlers:

- a. Click **Window > Preferences**.
- b. In the Preferences window, click **EGL** and ensure that **EGL support with JSF** and **EGL support with JSF Component Interfaces** are selected.
- c. Expand **EGL** and click **Generation**.
- d. On the Generation page, select the **Java** check box.
- e. Click **OK**.

Now JSF Handlers will generate automatically when you save them.

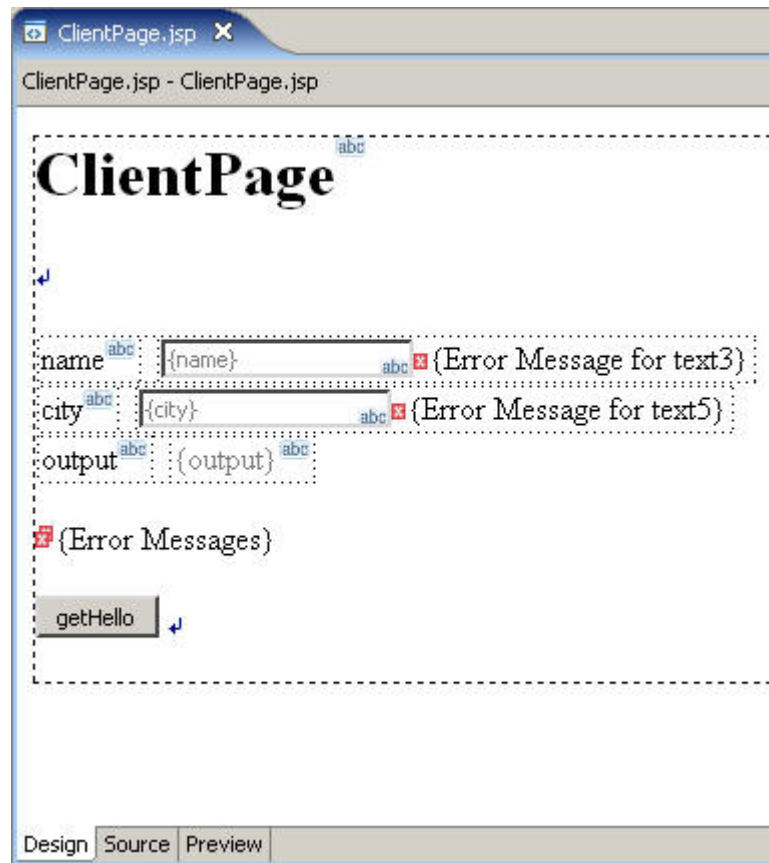
14. Open the ClientPage.jsp file in the editor. This page has fields on it based on the variables you created in the JSF Handler. These fields are pre-bound to the variables. This way, when the values of these fields change on the page, the variables will change to match. Similarly, when the values of the variables in the JSF Handler change, the values of the fields on the page will change to match.

This page also has several error message fields. This does not mean that your page has errors; these fields will show errors on the page if there are any when you run it.

15. Locate the Page Data view. If you cannot find this view, click **Window > Show View > Page Data**. This view shows the data available to your page, including the variables you created in the JSF Handler. It also shows functions in the JSF Handler.
16. In the Page Data view, expand **JSF Handler > Actions** and find the getHello() function.
17. Drag the getHello() function from the Page Data view directly onto the bottom of the page. A button bound to the function appears on the page. When the user presses the button on the page at run time, the function in the JSF Handler will run.

18. Save the page.

The page looks like this:



You now have a web page ready to use the web service. In the next lesson you will set up the project to act as a client for the service through this web page.

Lesson 4: Set up the project as a client

Just like the service project, the client project uses an EGL deployment descriptor file. However, in the client project, the EGL deployment descriptor will hold information about where to find services. You will import the WSDL file you created from the service and EGL will add appropriate binding information to the EGL deployment descriptor so the client can find the service.

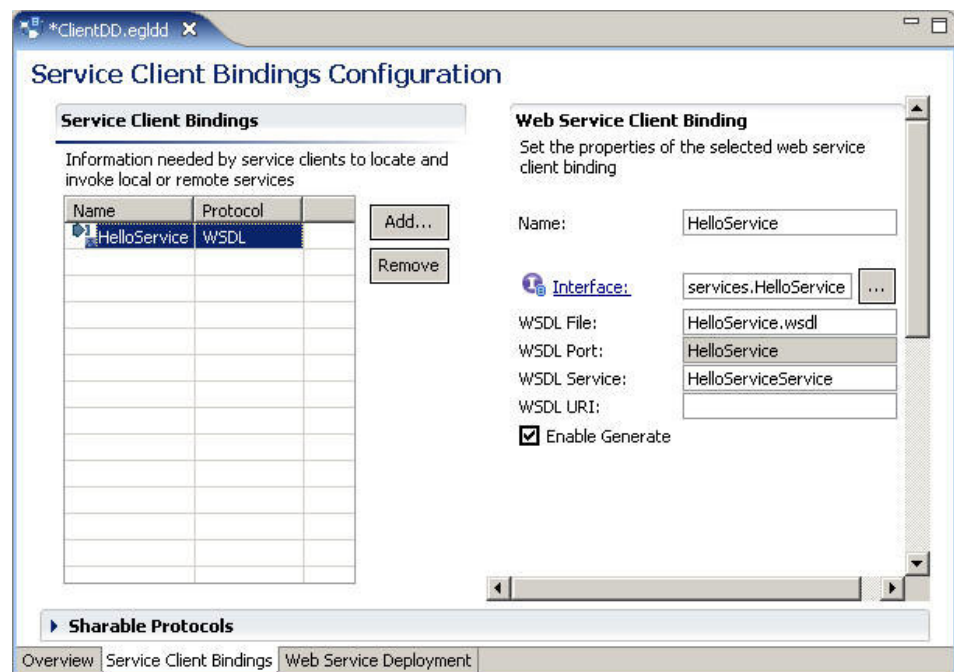
In the process, EGL creates an interface part to represent the service within the client project. Once the interface part is bound to the actual service, you can use that interface part as though it were the service itself.

Show Me

Set up the EGL deployment descriptor file

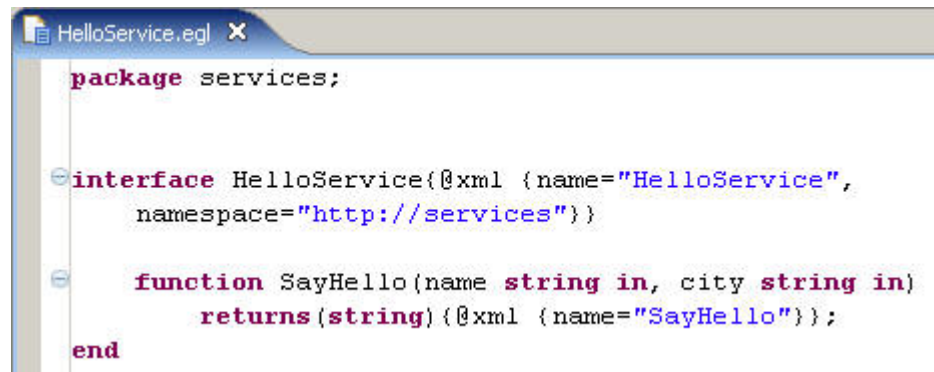
1. Open the client project's EGL deployment descriptor file by double-clicking `EGLClient/EGLSource/EGLClient.egldd` in the Project Explorer view. Note that the EGL deployment descriptor has no relation to the Java Deployment Descriptor folder, which is also in the `EGLClient` project.
2. In the deployment descriptor editor, go to the **Service Bindings** tab.

3. On the Service Bindings tab, click **Add**.
4. In the Add a Service Binding window, click **SOAP service binding**.
5. Click **Next**.
6. Select the **Choose wsdl file from workspace and copy it to the current project** check box.
7. Click **Browse**.
8. Select the HelloService.wsdl file, which is located in the EGLService project in the folder EGLSource\wsdl, and then click **OK**.
9. Under **Interface Options**, select the **Generate EGL Interface from WSDL file** radio button. With this option selected, EGL will create the parts you need to access the service automatically.
10. Accept the defaults for the other fields on the page and click **Next**. The New EGL Interface page shows a list of all the services described in the WSDL file. EGL will create an interface part in the client project for each service you select on this page. For now, only the service you created in the service project is listed here.
11. Make sure the check box for the HelloService service is selected and then click **Next**. The next page allows you to set where the new interface part will be created and what it will be named. You can also select which functions from the service to include in the interface part. By default, all of the functions in the service part are included.
12. In the **Source folder** field, make sure that the client project's source folder is specified: EGLClient\EGLSource.
13. In the **Package** field, make sure that the services package is specified.
14. Make sure that the check box next to the SayHello function is selected in the **Functions** list.
15. Click **Finish**. You have now created a service binding. Using this binding, the web page you created in this project can access the service. The binding in the deployment descriptor file looks like this:



16. Save and close the deployment descriptor file. Note that EGL has created an interface part in the project's services package. The interface part looks like

this:



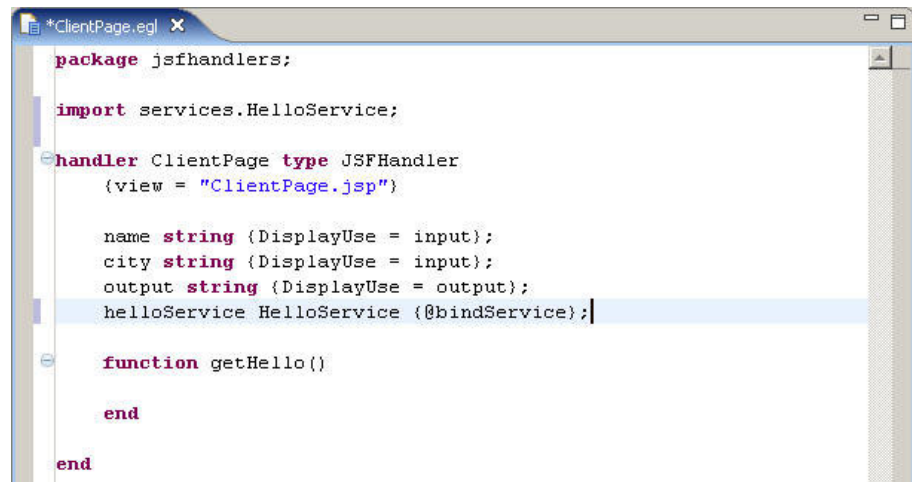
This interface is similar to the one you created in the service application, but this one has additional properties to refer to the binding in the deployment descriptor file.

17. Just as you did in the EGLService project, make sure the deployment descriptor appears in the client build descriptor file:
 - a. Double-click the build descriptor for the project to open it in the build parts editor. This file is named EGLClient.eglbld and is located in the project's EGLSource folder.
 - b. In the list of build descriptor options, the option named **deploymentDescriptor** should be set to EGLClient.
 - c. Click **OK** and close the file.

Use the service in the web page

1. Open the JSF Handler for the web page, named ClientPage.egl.
2. In the JSF Handler, create a variable from the interface part. It is usually easiest to use content assist to create a variable in this way:
 - a. In the JSF Handler file, place the cursor on a blank line immediately after the three variables you created in a previous section.
 - b. On the blank line, type the following code, as the first few characters of the interface part:
he
 - c. Press CTRL+space. The content assist completes the line for you with the following code:
helloService HelloService {@bindService};

Content assist also adds an **import** statement to the JSF Handler so you can use this part without specifying its complete location. The results look like the following picture:



```
package jsfhandlers;

import services.HelloService;

handler ClientPage type JSFHandler
{view = "ClientPage.jsp"}

    name string {DisplayUse = input};
    city string {DisplayUse = input};
    output string {DisplayUse = output};
    helloService HelloService {@bindService};

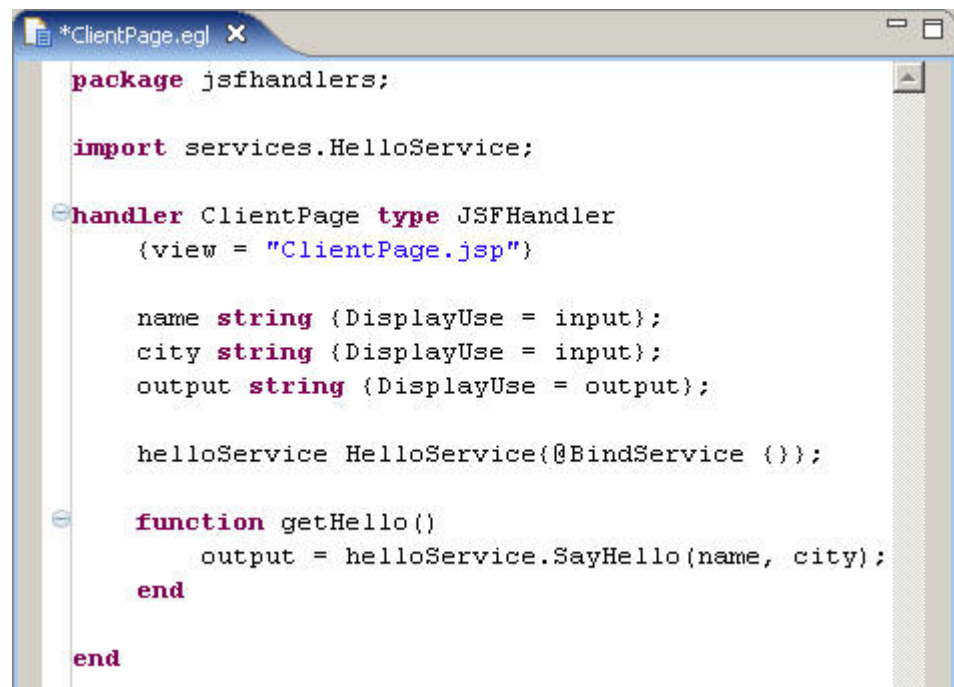
    function getHello()

    end

end
```

3. On a blank line within the getHello() function, invoke the SayHello() function in the service by passing it the name and city variables and assigning the output to the output variable:
`output = helloService.SayHello(name, city);`

Remember that you can use content assist by typing the first few characters of a keyword or part and then pressing CTRL+space. The JSF Handler looks like this:



```
package jsfhandlers;

import services.HelloService;

handler ClientPage type JSFHandler
{view = "ClientPage.jsp"}

    name string {DisplayUse = input};
    city string {DisplayUse = input};
    output string {DisplayUse = output};

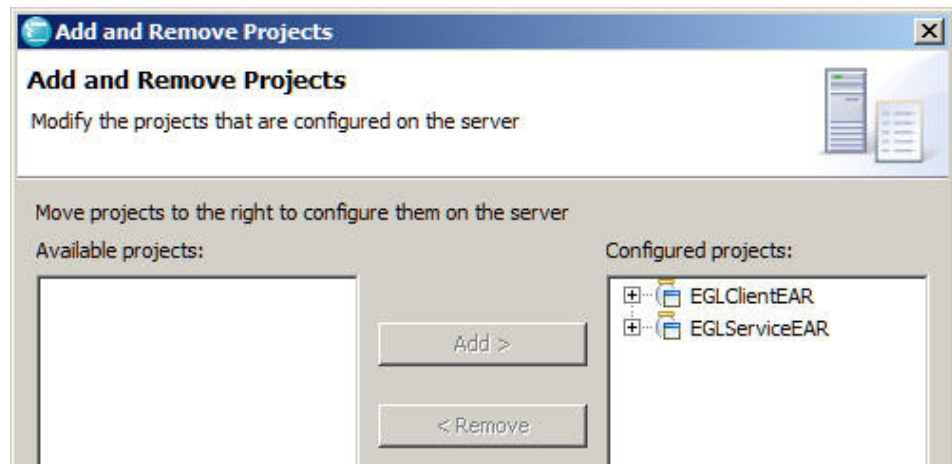
    helloService HelloService{@BindService {}};

    function getHello()
        output = helloService.SayHello(name, city);
    end

end
```

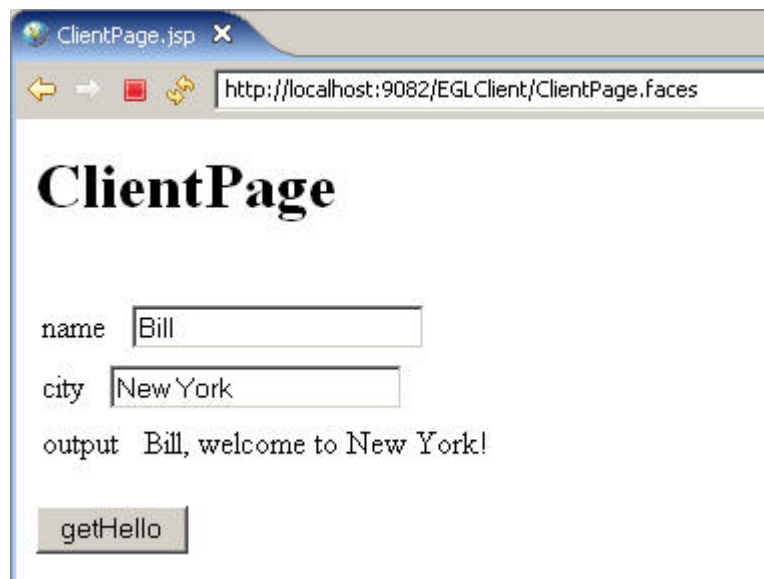
4. Save and close the JSF Handler.
5. Generate the entire client project by right-clicking it in the Project Explorer view and then clicking **Generate**.
6. Next, for testing purposes, you must let your application server know about the service you intend to call. In the Servers view, right-click the appropriate version of **WebSphere Application Server** and click **Add and Remove Projects**.

7. On the Add and Remove Projects page, verify that both the EGLClientEAR and the EGLServiceEAR are listed as **Configured projects**. If the EGLServiceEAR is in the **Available projects** column, click it to highlight it, then click **Add** and **Finish**. The window looks like the following picture:



Now the web page is ready to use.

8. In the Project Explorer view, right-click the ClientPage.jsp web page in the WebContent folder, not the JSF Handler, and then click **Run As > Run on Server**.
9. In the Define a New Server window, click the radio button for **Choose an existing server**, then click the appropriate version of **WebSphere Application Server**. Click **Finish**. The server publishes the page and displays it in the internal web browser. If you prefer to use an external web browser, you can copy the URL from the internal web browser and paste it into the URL field of the external web browser.
10. Type a name and city into the name and city fields, and then press the button on the page. The output field on the page shows a string such as "Bill, welcome to New York!" depending on the name and city you enter, as in this picture:



This may seem like a lot of work for a simple task, but these projects demonstrate how EGL can create services, clients, or both, and how those applications work

together. Using web services and clients in service-oriented architecture, you can integrate a wide variety of EGL and non-EGL applications in a way that is flexible and modular.

Summary

This is the end of the tutorial *Create a hello world service with EGL*.

The service you created and used in this tutorial was not powerful or complex, but it demonstrates how service-oriented architecture can separate your applications into modular, flexible pieces.

Lessons learned

By completing this tutorial, you learned how to do the following tasks:

- Create and configure an EGL project
- Create a web service with EGL
- Configure an EGL project to act as a service or as a client at run time
- Create a web page controlled by EGL
- Test an application on a web application server

You may want to continue learning by working with the tutorial application. Try adding additional functionality on your own. If you have done any other EGL tutorials, you could try putting the logic from those tutorials into a service, or you could try adding additional functions to your existing service.

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. enter the year or year, year.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.html>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds 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, or service names may be trademarks or service marks of others.



Printed in USA