

Rational Business Developer



Tutorial: Introducing EGL, a quick-start guide

Version 8.5

Rational Business Developer



Tutorial: Introducing EGL, a quick-start guide

Version 8.5

Note

Before using this information and the product it supports, read the information in “Notices,” on page 53.

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

Introducing EGL 1

Introduction 1

Lesson 1: Setting up EGL 5

 Preparing your workspace. 8

Lesson 2: Create the projects and import the database 9

 Installing the server (WebSphere Application

 Server) 9

 Installing the server (Tomcat) 10

 Create the EGL web project (either server) . . . 12

 Import the database 15

 Lesson checkpoint 17

Lesson 3: Set up the database connection 18

 Creating the connection 19

 Lesson checkpoint 23

Lesson 4: Create parts to access a database . . . 24

 Create parts from the database connection . . . 24

 Lesson checkpoint 26

Lesson 5: Create a web page. 27

 Create the JSP file from a template 28

 Preview the web page on the server 30

Lesson 6: Add data to the page. 33

 Add a record array to the Page Data view and

 the JSF Handler 33

 Display the data on the web page 35

 Call a function from the EGL library 37

 Test the page 40

Lesson 7: Pass a parameter to another page. . . . 41

 Add the link to allcustomers.jsp 41

 Add the parameter to the link 42

Lesson 8: Create an update page 44

 Create the updatecustomer.jsp file 44

 Add an EGL record and display it on the page 44

 Retrieve the data 46

 Update the record in the database 48

 Test the finished site 50

 You have completed the tutorial 50

Summary 50

Resources 51

 Completed allcustomers.egl file after lesson 6 . . 51

 Completed updatecustomer.egl file after lesson 8 52

Appendix. Notices 53

Trademarks 55

Introducing EGL

In this tutorial, you will learn how to build a simple dynamic web site using EGL. This site has two pages: one to display a list of records in a database and another to allow users to change the data from one of those records.

Enterprise Generation Language (EGL) is a development environment and programming language that you can use to write full-function applications quickly, freeing you to focus on the business problem your code is addressing rather than on software technologies.

Learning objectives

In this tutorial, you learn how to complete these tasks:

- Create and configure an EGL project
- Create EGL source code that accesses a data source
- Create two simple web pages that access data in a relational database
- Pass a parameter from one web page to another
- Test an application on a web application server

Time required

90 minutes

Introduction

In this tutorial, you will learn how to build a simple dynamic web site using EGL. This site has two pages: one to display a list of records in a database and another to allow users to change the data from one of those records.

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

Enterprise Generation Language (EGL) is a development environment and programming language that you can use to write full-function applications quickly, freeing you to focus on the business problem your code is addressing rather than on software technologies.

Learning objectives

In this tutorial, you learn how to complete these tasks:

- Create and configure an EGL project
- Create EGL source code that accesses a data source
- Create two simple web pages that access data in a relational database
- Pass a parameter from one web page to another
- 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 dynamic web sites while working on the tutorial, it could take longer to finish.

Skill level

Introductory

System requirements

To complete this tutorial, you need to have the following tools and components installed:

- Enterprise Generation Language (EGL)
- Either WebSphere® Application Server or Apache Tomcat server. The instructions give you the option of using either server; and they will include instructions for installing Apache Tomcat if you do not have WebSphere Application Server.

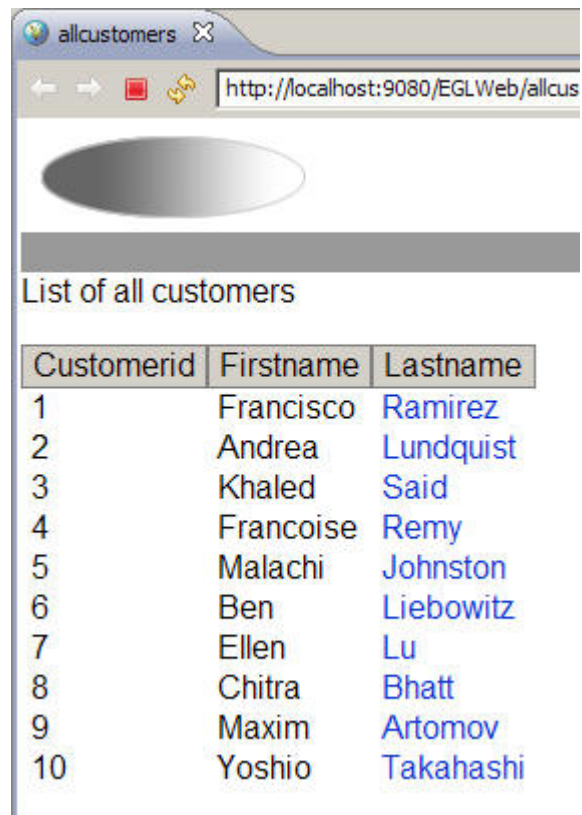
Prerequisites

You will be best prepared to complete this tutorial if you have programmed in any third- or fourth-generation language such as COBOL, RPG, or a client/server language, and if you are familiar with these topics:

- Terms used with relational databases, such as table, row, and column
- Basic web-related terms such as browser, web page, and web application server

Tutorial application

In this tutorial, you will create an EGL web project and import a sample database. You will then create a simple EGL web application that works with this database. The first of the two web pages in your application shows a list of customers from data stored in the database:



The screenshot shows a web browser window with the title 'allcustomers'. The address bar displays 'http://localhost:9080/EGLWeb/allcus'. Below the address bar is a large, empty, rounded rectangular area. Underneath this area is a horizontal bar, and then the text 'List of all customers'. Below the text is a table with three columns: 'Customerid', 'Firstname', and 'Lastname'. The table contains 10 rows of customer data. The 'Lastname' column contains blue, underlined text, suggesting it is a hyperlink.

Customerid	Firstname	Lastname
1	Francisco	Ramirez
2	Andrea	Lundquist
3	Khaled	Said
4	Francoise	Remy
5	Malachi	Johnston
6	Ben	Liebowitz
7	Ellen	Lu
8	Chitra	Bhatt
9	Maxim	Artomov
10	Yoshio	Takahashi

The second web page shows details about one customer and allows users to change those details:

updatecustomer

http://localhost:9080/EGLWeb/update

Update Customer Information

Customerid: 1

Firstname: Francisco

Lastname: Ramirez

Password: pa55word

Phone: (201)652-3456

Emailaddress: ramirez@abc.ibm.com

Street: 14 Maple Avenue

Apartment:

City: Wake Forest

State: NJ

Postalcode: 07542

Directions:

Update this record

EGL is the language that you use to manage the interaction between users and the database:

- After retrieving data from the database, the functions that you write in EGL can apply business rules as appropriate.
- When preparing to present data to users, those functions can alter characteristics of the web-page display, even choosing which page to present.
- On accepting the users' responses, those functions can apply additional business rules before storing the data.

Each of the two pages pictured above are controlled by EGL logic parts called *Handlers*, which control the runtime interaction with a user interface. In this case, the Handler parts are *JSF Handler parts*, Handler parts specialized to control a single web page at run time. A JSF handler's function is invoked by a user click, and the function in turn invokes a library function that you create. The result is that a user working at a web browser can view and alter data stored in a database.

As shown in this tutorial, EGL promotes code reuse in several ways:

- First, EGL lets you define *DataItem parts*, which are a simple type of EGL data structure. A *DataItem part* is based on a single primitive data type, with any number of added EGL properties. For example, if your application uses many telephone numbers, you can define a *DataItem* to represent a telephone number. This *DataItem* would use a numeric primitive as its base and have properties that define its exact length and output formatting. You can then create many variables or other data parts in your code based on that single *DataItem part*. *DataItem parts* are similar to entries in a data dictionary, with each part including details on data size, type, formatting rules, input-validation rules, and display suggestions. You define a *DataItem* once and can use it as the basis for any number of variables or record fields.
- Second, EGL lets you define *Record parts*, which are used as the basis for structured data. A *Record part* is a collection of other data parts (such as *DataItem parts* or primitives) that are organized into a hierarchical structure. This kind of data part is often used to create variables that access a file or relational database.

In this tutorial, you create a record part that represents contact information for a customer. This *Record part* contains data items representing information about a customer, such as first and last name, telephone number, and address. Also, this *Record part* is specialized, or *stereotyped*, as an *sqlRecord part*, to work directly with the database.

A *Record part* can reference a series of *DataItem parts*, as shown in this tutorial. If you organize your data in this way, you can realize a more consistent definition of your data parts and can increase efficiency over time. Your changes to a single *DataItem part* will cause a change in every variable that accesses the related, stored data.

- Third, EGL lets you create *source libraries*, which contain functions, data parts, and constants that provide a basis for logic reuse and for modular programming based on proven code.

EGL also provides the Data Access Application wizard, which you will use to create the elementary code necessary to access a relational database. This wizard creates EGL parts that have these specific purposes:

- *Record parts* that reflect the characteristics of each database table.
- *DataItem parts* that reflect the characteristics of each table column.
- *Source libraries* that include functions to create, read, update, and delete rows in the database.

The library functions include parameters that are based on the *Record parts* created by the wizard. You can start to build a robust application just by invoking those functions with arguments that are based on the same *Record parts*.

Lesson 1: Setting up EGL

In this lesson, you will prepare to use EGL by setting up your workspace and enabling the EGL capability.

Before you can begin this tutorial, you must make sure that your system is configured to use EGL. You will need to do the steps in this lesson only once, even if you create many EGL projects. These steps make sure that EGL is installed and enabled on your system.

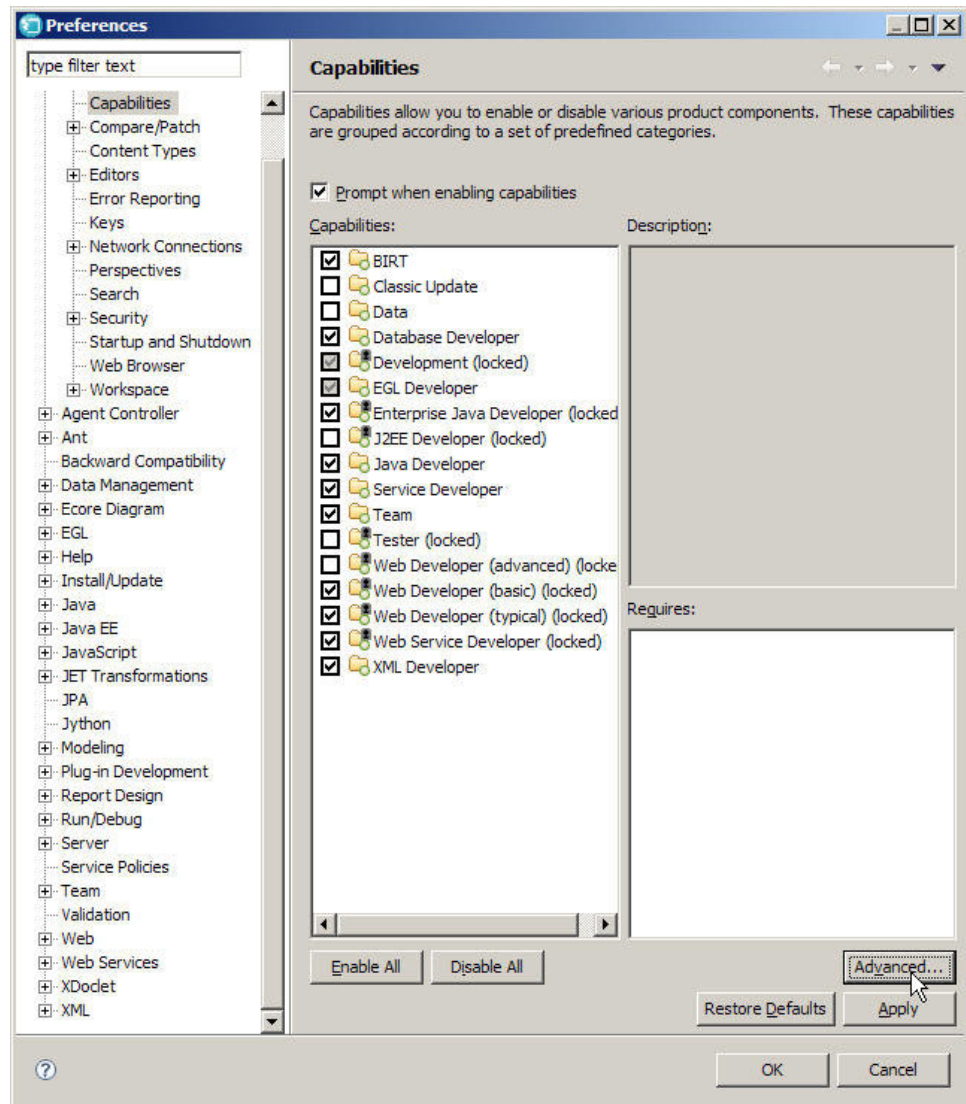
The workbench hides options that you are not using, based on which capabilities are enabled. For example, when the EGL Development capability is disabled,

EGL-related projects and file types do not appear in the **File > New** menu. In this way, capabilities keep the workbench from becoming cluttered with too many options. For more information about EGL capabilities, see the help topic Enabling EGL capabilities.

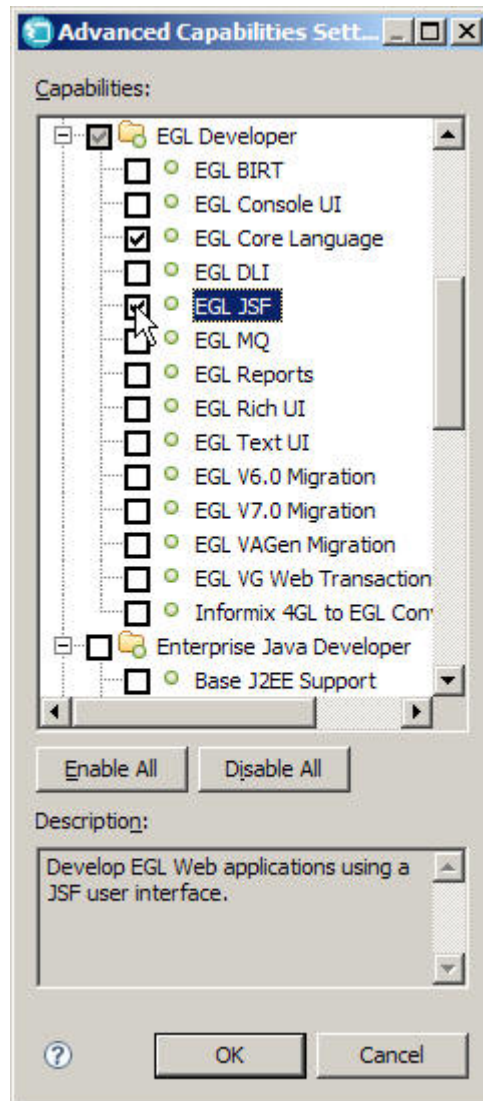
Follow these steps to enable EGL:

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. From the menu bar, click **Window > Preferences**. The Preferences window opens.
3. At the left side of the Preferences window, expand **General** and click **Capabilities**.
4. On the Capabilities list, click **Advanced**. The **Capabilities** page looks like the following example. You may have other capabilities available, depending on the products and options you have installed.



5. In the Advanced Capabilities Settings dialog, expand **EGL Developer** and select the check boxes for all the EGL capabilities you need. For this tutorial you need only the **EGL Core Language** and **EGL JSF** capabilities.



6. Click **OK**.

You have enabled the necessary EGL capabilities to create EGL-related files and projects. There might be many other capabilities available on the capabilities page. You do not need to enable any other capabilities for this tutorial, but you other tasks may require other capabilities.

Preparing your workspace

To follow this tutorial easily, open the web perspective and close the Welcome page and any open files from other projects. To open the web perspective, follow these steps:

1. Close the Welcome if it is open.
2. In the workbench, click **Window > Open Perspective > Other**. The Open Perspective window opens.
3. Click **Web**.
4. Click **OK**.

The web perspective loads in the workbench, showing the views that you need to complete this tutorial. You can go to any other perspective by clicking the **Open a perspective** button:



You can also click **Window > Open Perspective > Other** and click a perspective. If at any time you have closed or resized the views, you can click **Window > Reset Perspective** to restore the perspective to its defaults.

Lesson 2: Create the projects and import the database

In this lesson, you will create projects to hold your EGL application and add a database to use.

In this tutorial, you will spend most of your time working with files in an EGL web project. This project will contain the EGL code, web pages, and sample database that make up the logic, interface, and data for the application. Before you begin creating projects, however, you need to decide which server you will use in the tutorial.

If you choose WebSphere Application Server, you will need an Enterprise Application Resource project (EAR project) in addition to the EGL web project. The EAR project contains information about deploying an application in the J2EE framework, including how to run it on a server and how to connect it to data sources. An EAR project can contain one or more other projects, meaning that the EAR project contains information on deploying those projects. The projects contained by an EAR project are called *modules*. In this case, your EGL web project will be a module within the EAR project. In a large application, an EAR project could have many different types of modules doing different jobs.

If you choose Apache Tomcat, however, you will not need an EAR project, and in fact, you will not be able to use an EAR project. WebSphere Application Server is a full-featured application server that can run each of the types of modules in the J2EE framework, including web projects (like your EGL web project), EAR projects, and Enterprise JavaBean (EJB) projects. On the other hand, Tomcat is a web server, designed to run only web projects; it does not support other types of J2EE projects such as EAR projects or EJB projects.

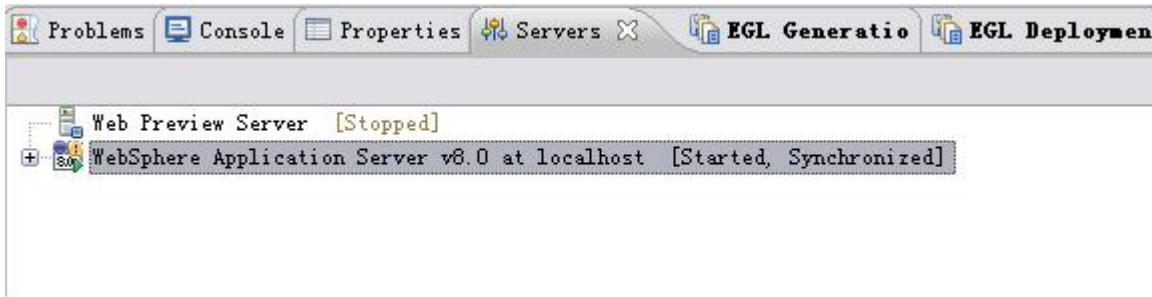
The tutorial application will run the same way regardless of which server you use, but you should be aware of the differences between the servers. If you use WebSphere Application Server, you will create an EAR project and put the database connection information in that EAR project. If you use Tomcat, you will put the database connection information directly into the EGL web project.

For the rest of the tutorial, be aware of sections that apply to WebSphere Application Server or to Apache Tomcat. If the instructions refer to projects, files, or options that you do not have, check to see that you are in the correct section for your server.

Installing the server (WebSphere Application Server)

WebSphere Application Server typically requires a separate installation process from your EGL product. You must install WebSphere Application Server into the same package group that contains your EGL product. If you can see WebSphere Application Server in the **Servers** view, located by default at the bottom of the

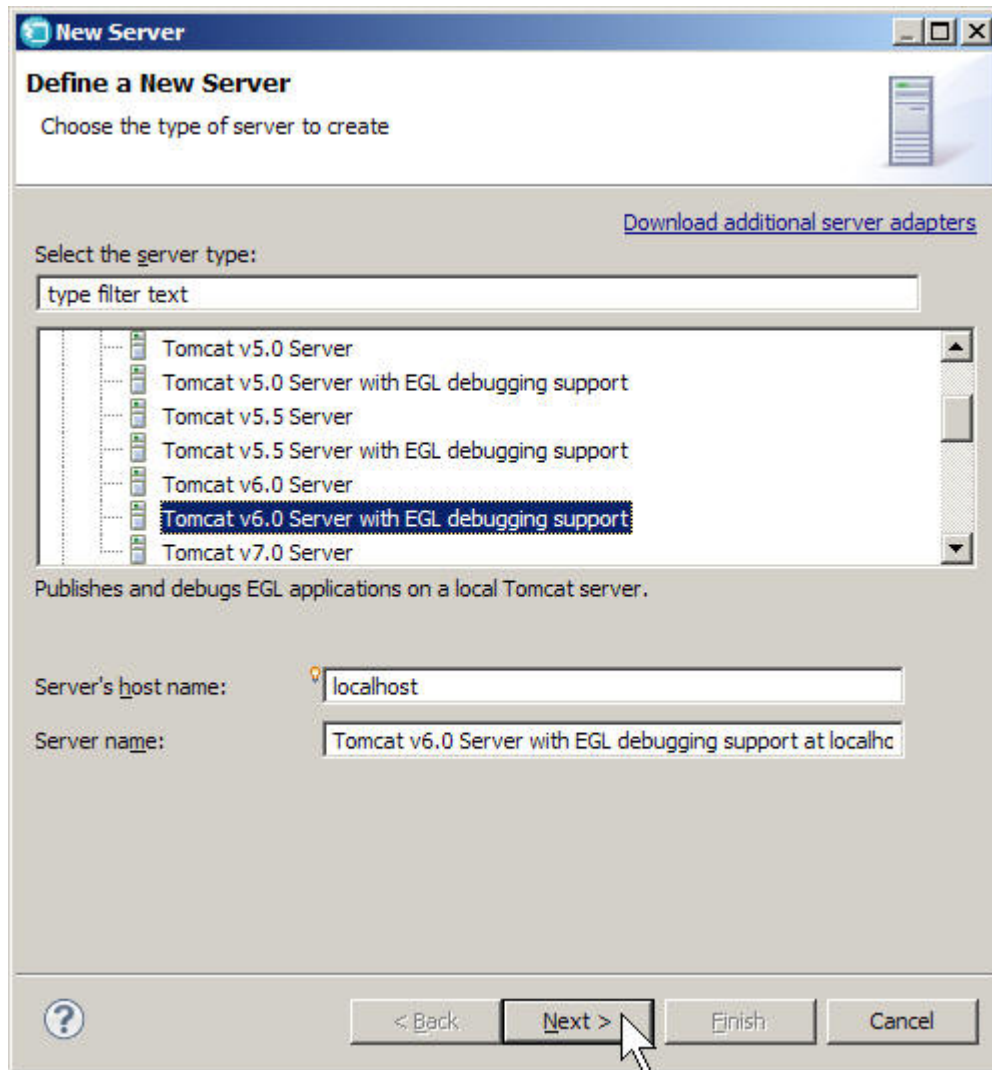
workbench, then the product is installed. If not, you must buy the product or use Tomcat instead.



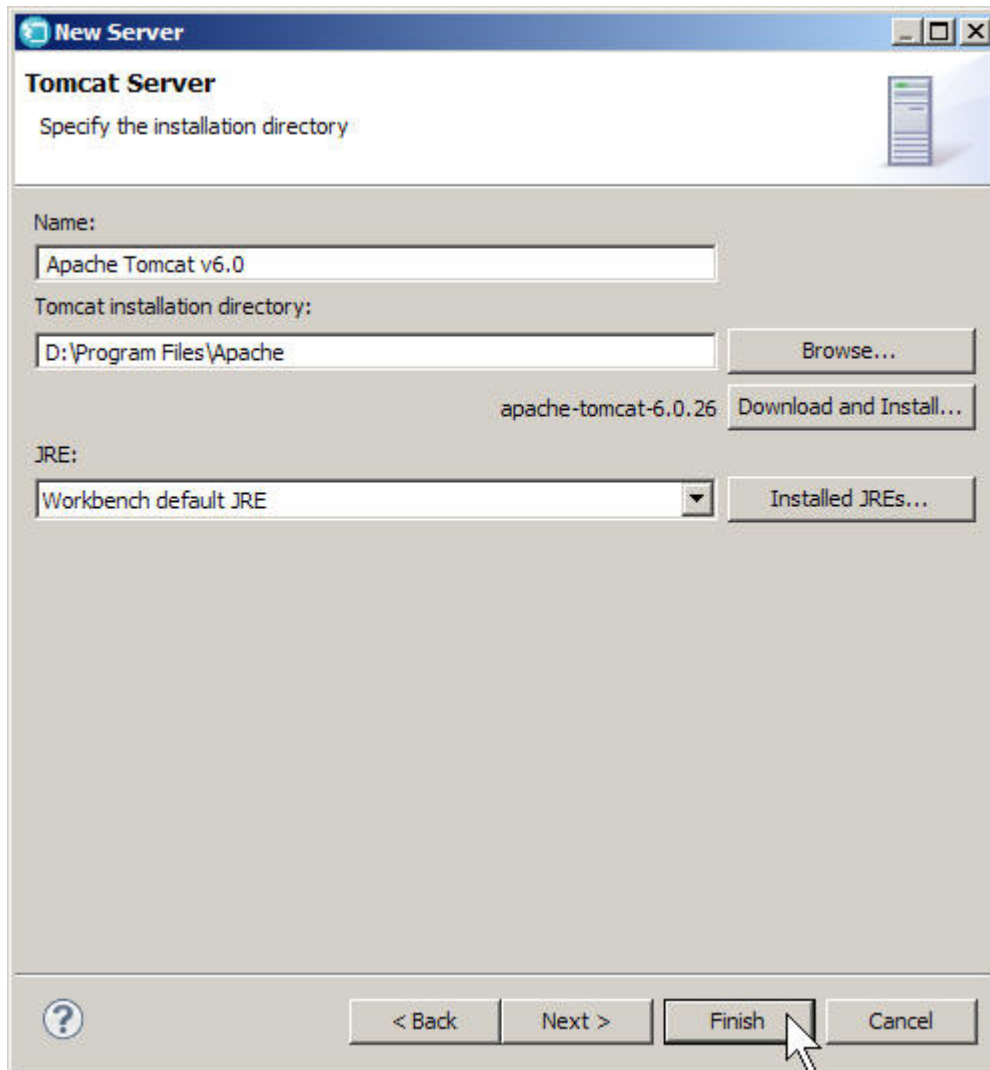
Installing the server (Tomcat)

Installing Tomcat is usually as easy as downloading and unzipping the server files and then telling the workbench where to find them. Follow these steps to install and configure Tomcat 6.0:

1. In the **Servers** view, located by default at the bottom of the workbench, right-click anywhere in the blank space, then click **New > Server**.
2. In the Define a New Server window, expand IBM if necessary and select the version of Tomcat that is installed on your system, or that you wish EGL to install on your system for you. If you select a version with debug support, you can step through your programs in the EGL debugger.



3. In the Tomcat Server window, you have an option:
 - If you have already installed Tomcat to your system, browse to the **Tomcat installation directory**.
 - If you have not yet installed Tomcat, click **Download and Install** and follow the instructions on your screen. Repeat this step when you have installed the software.



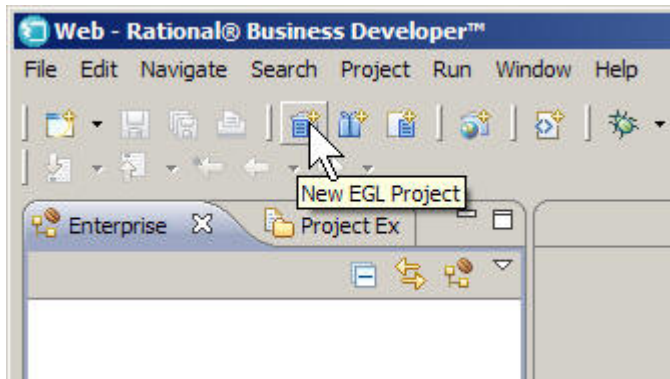
4. Click **Finish**. The new server is displayed in the **Servers** window.
5. Leave the EGL Workbench for a moment and use Windows Explorer to browse to the folder where you installed Tomcat. Look for the `lib` folder inside this folder. For example, the path might look like this:
`D:\Program Files\Apache\lib`
 Make sure this folder contains a file named `derby.jar`. If not, copy the file into the `lib` folder from the following folder in your installation directory:
`installation_directory\plugins\org.apache.derby.core_10.1.2.1`

Create the EGL web project (either server)

Because your project will have a web page interface, you need to create an EGL web project. An EGL web project combines the features of a dynamic web project and an EGL project. If you are using WebSphere Application Server, you will also create an EAR project.

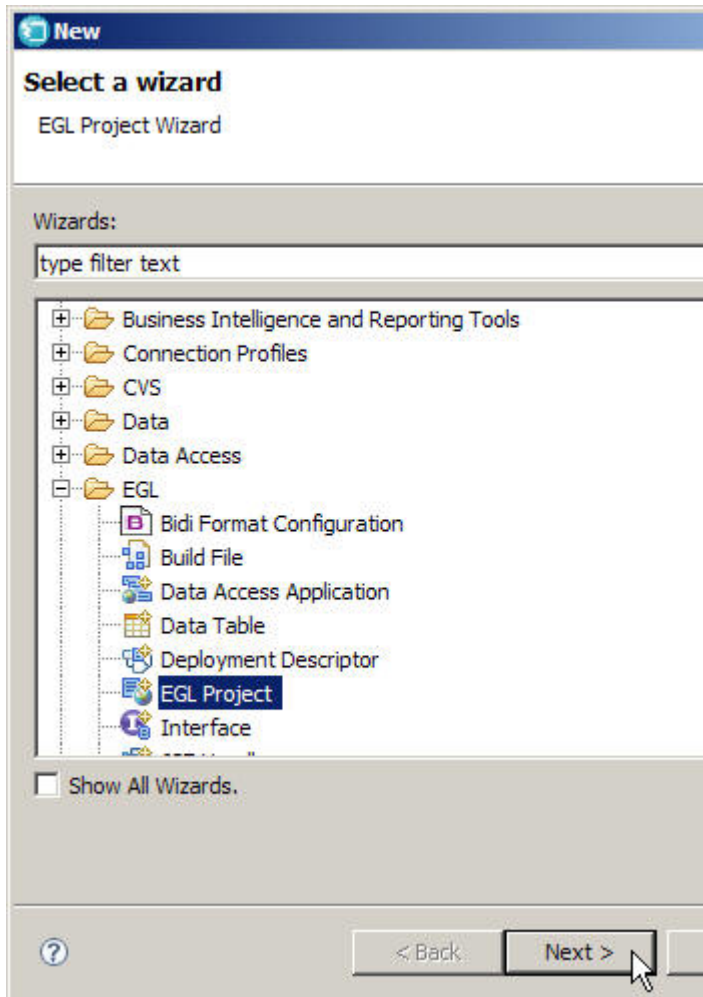
Show Me

1. Click the New EGL Project icon in the top left of the workbench.

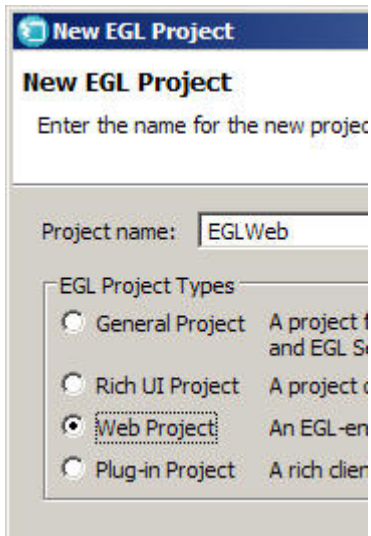


EGL displays the New EGL Project wizard.

2. Alternatively, you can click **File > New > Other**, then in the Select a Wizard dialog, expand the **EGL** folder and click **EGL Project**, then click **Next**. EGL displays the New EGL Project wizard.



3. In the **Project name** field, enter this name for your project:
EGLWeb
4. Under **EGL Project Types**, click **Web Project**. Click **Next**.



5. Click **Next**.
6. In the **Target Runtime** field, select the name of the application server that you verified at the beginning of this lesson.
7. Under **Build Descriptor Options**, make sure that **Create a new build descriptor** is selected.
8. Select the **Show Advanced Settings** check box and click **Next**.
9. Leave the **Use the default location for the project** check box selected.
10. If you are using WebSphere Application Server, the **Add project to an EAR** check box should be selected, and the name EGLWebEAR should appear for the **EAR project name**. Accept both of these defaults.
11. You do not need to change anything in the **Modify project facets** section.
12. Clear the **Create an EGL deployment descriptor** check box. EGL deployment descriptor files contain information on deploying and using web services. This tutorial does not connect to any web services.

New EGL Project

New EGL Web Project

Specify the advanced settings for the EGL Web Project

Project location

By default, the project is located within the Eclipse workspace.

☒ Use the default location for the project

Directory:

EAR Membership

You can add this project to an enterprise application resource package (EAR), such as database connections.

☒ Add project to an EAR

EAR project name:

Modify project facets

Click to modify the facets that will be applied to this project

EGL Project Features Choices

☐ Create an EGL service deployment descriptor

☐ EGL with BIRT report support

☐ EGL with low-level MQ API support

☐ EGL with LDAP support

☐ EGL with i5/OS objects support

13. Click **Finish**.
14. You may see a message asking if you want to switch to the J2EE perspective. If you see this message, click **No**.

The new project or projects are created in your workspace. The workbench may display a Technology Quickstarts window with Help information. You can close this window.

Import the database

This tutorial includes a sample Derby database to be used in your application. In these steps, you add this database to your project. For more information on Apache Derby, an open-source relational database, see <http://db.apache.org/derby/>.

1. Click the following link and download the sample database to a temporary folder on your computer, such as your desktop:

Sample database

It does not matter where you save the database, as long as you can find it again later.

Alternately, you can find this sample database in your product installation directory in the following location:

`shared_resources/plugins/com.ibm.etools.egl.tutorial0001.doc_version/
resources/EGLDerbyDB.zip`

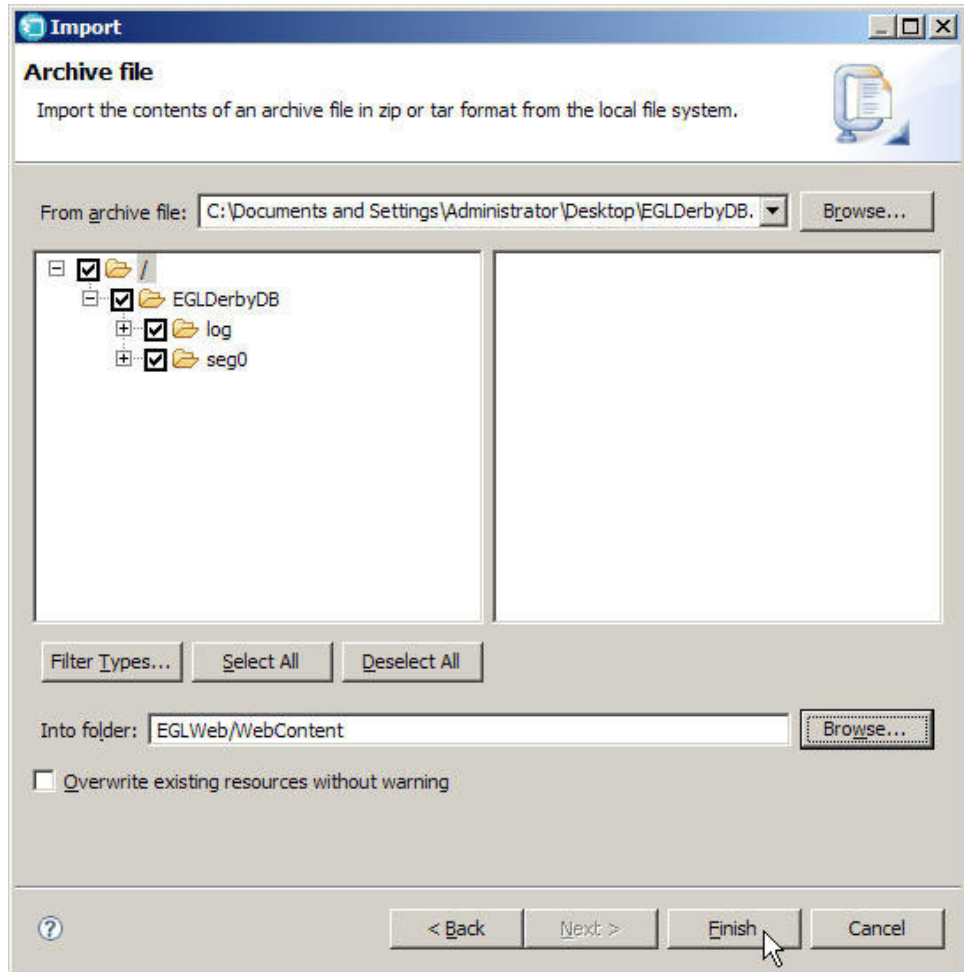
shared_resources

The shared resources directory for your product, such as C:\Program Files\IBM\SDP70Shared on a Windows system or /opt/IBM/SDP70Shared on a Linux system. If you installed and kept a previous version of an IBM® product containing EGL before installing your current product, you may need to specify the shared resources directory that was set up in the earlier installation.

version

The installed version of the plugin. If more than one is present, use the one with the most recent version number, unless you have a reason to use an older version.

2. In the workbench, click **File > Import**.
3. In the Import window, expand **General**, click **Archive File**, then click **Next**.
4. In the Archive file window, **From archive file** field, enter the location of the file you just downloaded. You can use the **Browse** button to find it.
5. At the bottom of the wizard, next to the **Into folder** field, click the **Browse** button.
6. In the **Import into folder** window, expand **EGLWeb**, click the **WebContent** folder to select it, then click **OK**. This folder is where the database will be added to your project. The Import window looks like this:



7. Click **Finish**.

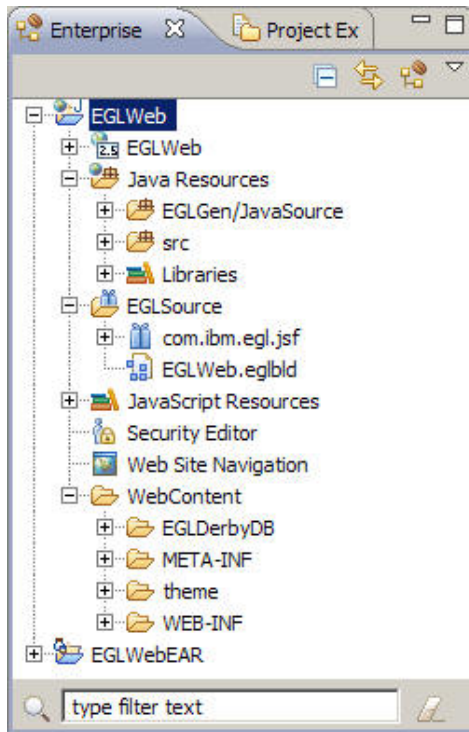
The database is added to your workspace in the WebContent folder of the EGLWeb project. Do not edit any of the files in the database directly. Later, you will create an EGL application to view and edit this database.

Lesson checkpoint

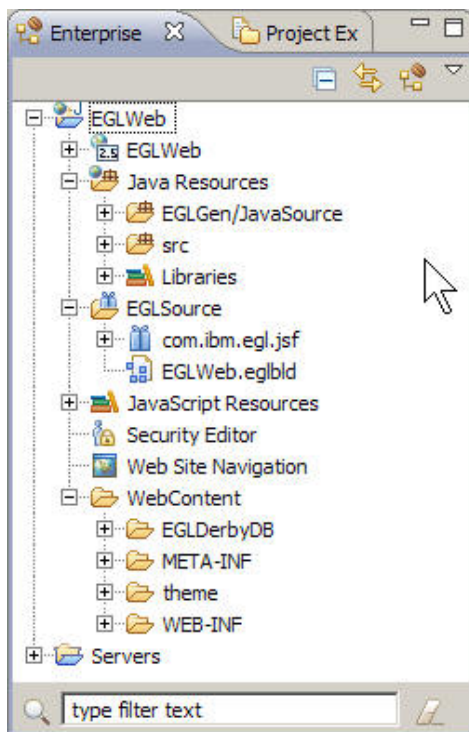
In this lesson, you created one or two projects, depending on your server. You can explore the project or projects in the Project Explorer view:

- The EGLWeb project will contain the EGL code, web pages, and other files associated with the application. In particular, you will work with the EGLSource and WebContent folders which will contain the EGL source code files and the web pages for the application, respectively.
- The EGLWebEAR project is the Enterprise Application Resource for the EGLWeb project. You have this project only if you are using WebSphere Application Server.

The Enterprise Explorer view looks like this if you are using WebSphere Application Server:



The Enterprise Explorer view looks like this if you are using Tomcat:



Lesson 3: Set up the database connection

In this lesson, you will connect your project to the database that you imported in the previous lesson.

Because the goal of this tutorial is not to teach you how to connect to a database, this lesson will not explain the process in detail. In short, you will set up a database connection that allows your EGL application to connect to the database both when you design the application (the design-time connection) and when you run the application on the server (the run-time connection).

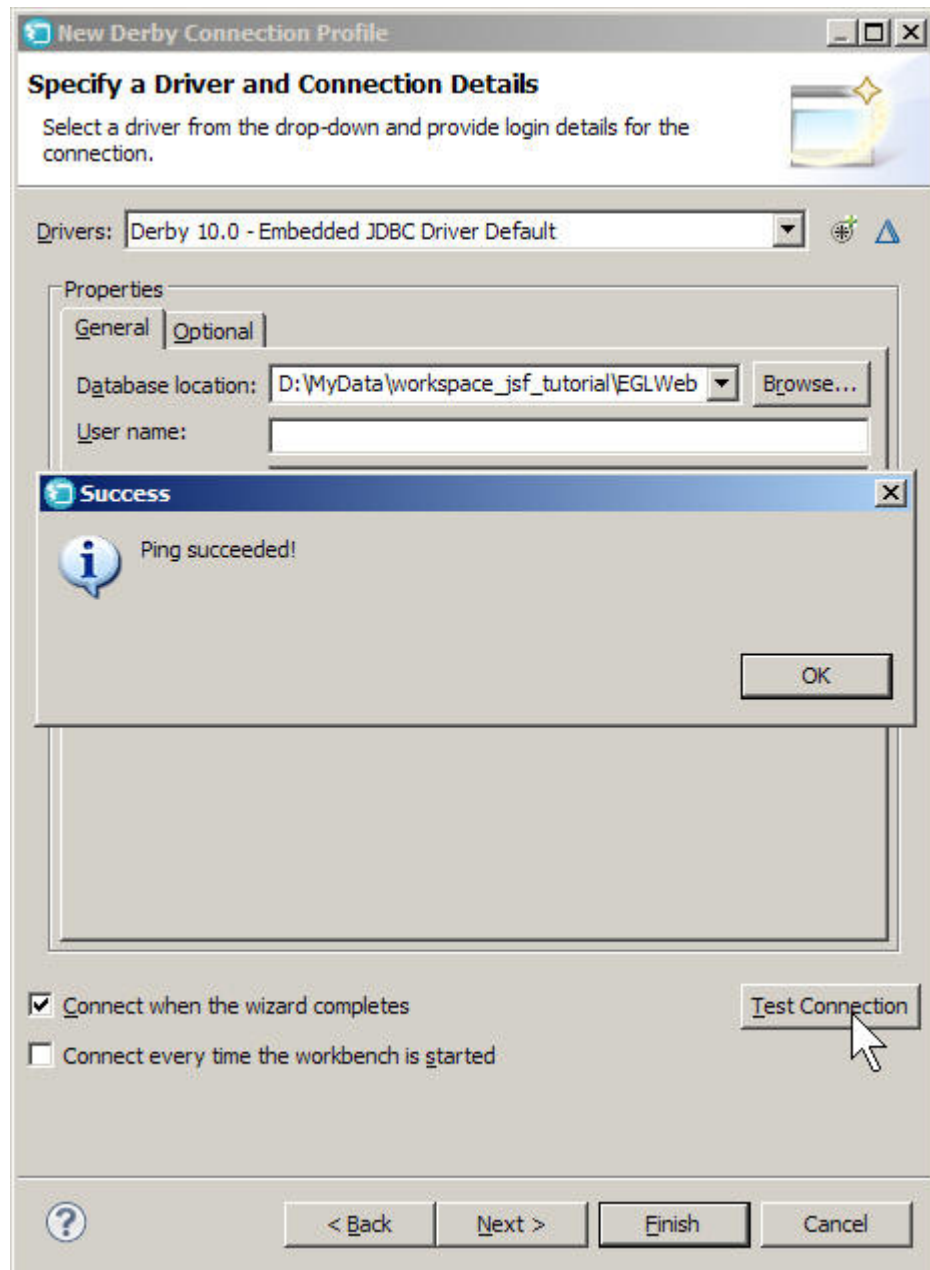
As explained in a previous lesson, WebSphere Application Server uses database connection information in the EAR project. Tomcat uses database connection information in the web project. Thus, the database connection steps differ for each server.

Creating the connection

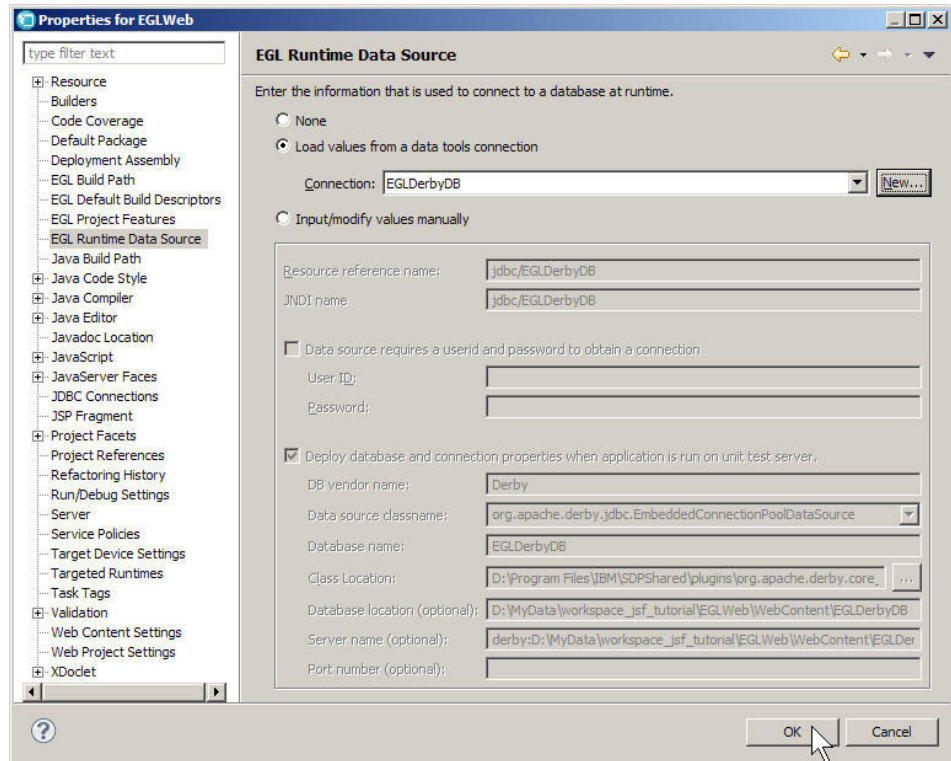
Regardless of which server you are using, you must create a design-time connection to the database. If you are using WebSphere Application Server, EGL automatically creates a matching run-time connection as well. If you are using Tomcat, additional steps later in this lesson will guide you through associating this connection with Tomcat. In the next lesson, you will also use this design-time connection to create starter EGL code.

Show Me

1. In the Enterprise Explorer view, right-click the EGLWeb project and then click **Properties**.
2. In the Properties window, click **EGL Runtime Data Source**.
3. On the EGL Runtime Data Source page, select **Load values from a data tools connection** and then click **New**. The New Connection window opens.
4. In the Connection Profile window, under **Connection profile types**, click **Derby**.
5. For **Name**, type the following name:
EGLDerbyDB
6. You can leave the description blank. Click **Next**.
7. From the **Drivers** list, leave the default **BIRT SampleDb Derby Embedded Driver**.
8. Under **Properties**, in the **Database location** field, click **Browse** and navigate to the following folder:
workspace-location/EGLWeb/WebContent/EGLDerbyDB
workspace-location is the full path to your current workspace. Click **OK**.
9. Clear the **User name** field and leave the **Password** field blank. You do not need a user name or password for this database.
10. Accept the default value for **URL**. You can clear the check box for **Create database (if required)** as the database already exists. Make sure **Connect when the wizard completes** is selected.
11. Click **Test Connection**. If all information is correct, the New Connection window should look like the following example, with your own workspace and location information in the **Database location** field:

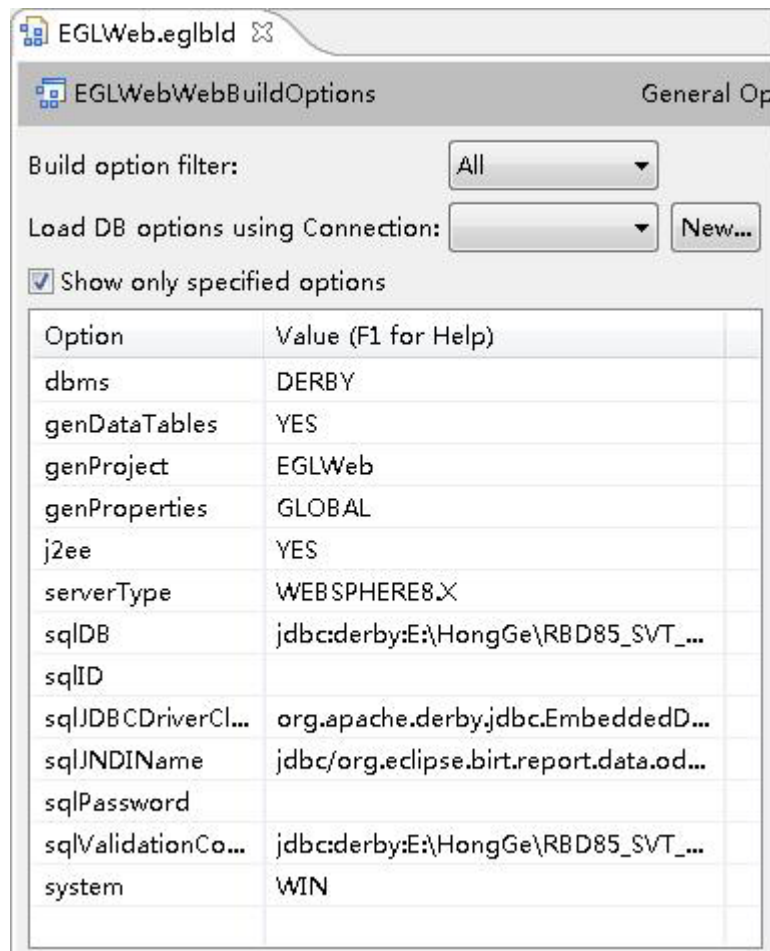


12. Click **OK** to close the test connection window.
13. Click **Finish**. The new connection is created and the necessary information for the connection is filled into the fields below:



Note that EGL has given this connection a *JNDI name*, which is an identifier for the connection. By default, the JNDI name is `jdbc/EGLDerbyDB`, based on the name of the database. The application will use this name to access the database connection at run time.

14. Click **OK**.
15. You may see a window asking if you want to update the information in the default build options for this project. If you see this window, click **Yes**.
16. In the Project Explorer view, expand the `EGLWeb` project and the `EGLSource` folder. Open the build file for the project by double-clicking the `EGLWeb.eglbld` file in the Project Explorer view. The build file opens in the build parts editor.
17. Check to see that the EGL Runtime Data Source window has set the build descriptor options based on the connection information. The build descriptor options should look like this for WebSphere Application Server:



For the database connection to work, the following options need to be set:

dbms

This build descriptor option indicates the type of database, in this case DERBY.

sqlDB

This build descriptor option indicates the *connection URL*, or a string that the server uses to find the database. The format of the connection URL differs for each type of database, but for Derby, the format is the connection protocol (in this case JDBC), a colon separator, the type of database (Derby), another colon separator, the path to the database on disk, and any parameters for the connection. In this case, the connection URL is something like the following example, with the path to your database in place of D:\MyData\workspace_jsf_tutorial:

```
jdbc:derby:D:\MyData\workspace_jsf_tutorial\EGLWeb\WebContent\EGLDerbyDB
```

sqlValidationConnectionURL

This build descriptor option sets a connection URL to be used to validate the connection to the database. In this case, as in most cases, this option is the same as **sqlDB**.

sqlJDBCDriverClass

This build descriptor option sets the name of the *database driver*, the program used to access the database. The New Connection window retrieved this name from the derby.jar file:
org.apache.derby.jdbc.EmbeddedDriver.

sqlJNDIName

The JNDI name that represents the connection at run time.

18. If the build descriptor options have been set based on the information you filled into the New Connection window, close the build descriptor without making any changes. If the build descriptor options have not been set, follow these steps to set them:

- a. In the **Load DB options using Connection** list, select your EGLDerbyDB connection. Several of the options are set, except for the **sqlJNDIName** option.
- b. Set the **sqlJNDIName** option to the following JNDI name, exactly as shown:

jdbc/EGLDerbyDB

Note: To open the **sqlJNDIName** option for editing, click twice slowly in the **Value** column next to that option. Also, you can click three times quickly in the **Value** column.

The values of the build descriptor options now match those described above.

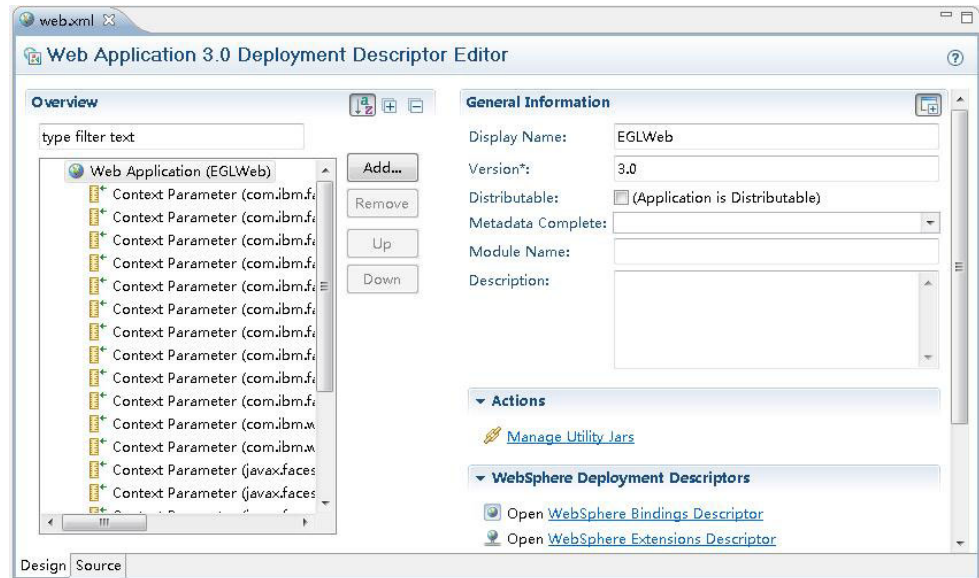
- c. Save and close the build descriptor.
- d. Optionally, set the EGL Runtime Database Connection window to make these changes in the future by enabling the associated preference. Click **Window > Preferences** and then click **EGL > Default Build Descriptor**. Under **Update default build descriptor options for project when runtime data source is modified**, select **Always** to update the build descriptor options automatically, or select **Prompt** to give you the option. This preference takes effect the next time you use the EGL Runtime Database Connection window.

Lesson checkpoint

In this lesson, you set up a database connection for the project.

When you used the EGL Runtime Data Source page of the project's Properties window, you first created a design-time connection to the database using the workbench's data tools. Then, if you were using WebSphere Application Server, EGL used the information in this design-time connection to create a matching connection to be used at run time. In this case, the changes EGL made to your projects include:

- EGL set the values of certain database-related build descriptor options, as explained earlier in the lesson.
- EGL created a *JNDI name* to use as a name for the connection. By default, the JNDI name created for your project is jdbc/EGLDerbyDB, based on the name of the database.
- EGL added a resource reference to that JNDI name in the EGLWeb project's web deployment descriptor, WebContent\WEB-INF\web.xml. Now the EGLWeb project can use the data source defined in the EAR project, using the JNDI name. The editor looks very different depending on the version of the application server you use. The following example shows the editor for WebSphere Application Server Version 8.



If you are using Tomcat, your project doesn't have an EAR project, so EGL added a contexts file to the web project that gives the information for the connection. The contexts file performs essentially the same task as the information in the deployment descriptors: it associates the JNDI name with the location of the database and other information that the server needs to connect to it. The connection is valid only for this project.

Note: From this point forward, most of the steps are the same regardless of which server you are using.

Lesson 4: Create parts to access a database

In this lesson, you will create the data and logic parts that allow you to access the sample database.

The EGL Data Access Application wizard creates the EGL code necessary to access a database. Although you can customize the code when the wizard runs, typically the wizard creates the following EGL parts:

- For each table you select from the database, the wizard creates a *record part* representing that table. This Record part is a series of fields, each representing the columns in the table.
- For each table you select from the database, the wizard creates a *library part*. This library contains EGL functions that you can use to read from or write to the database.
- For each row in the database tables you select, the wizard creates a *DataItem part*. These DataItem parts represent the columns in the database. The Record parts created by the wizard are made of a sequence of these DataItems.

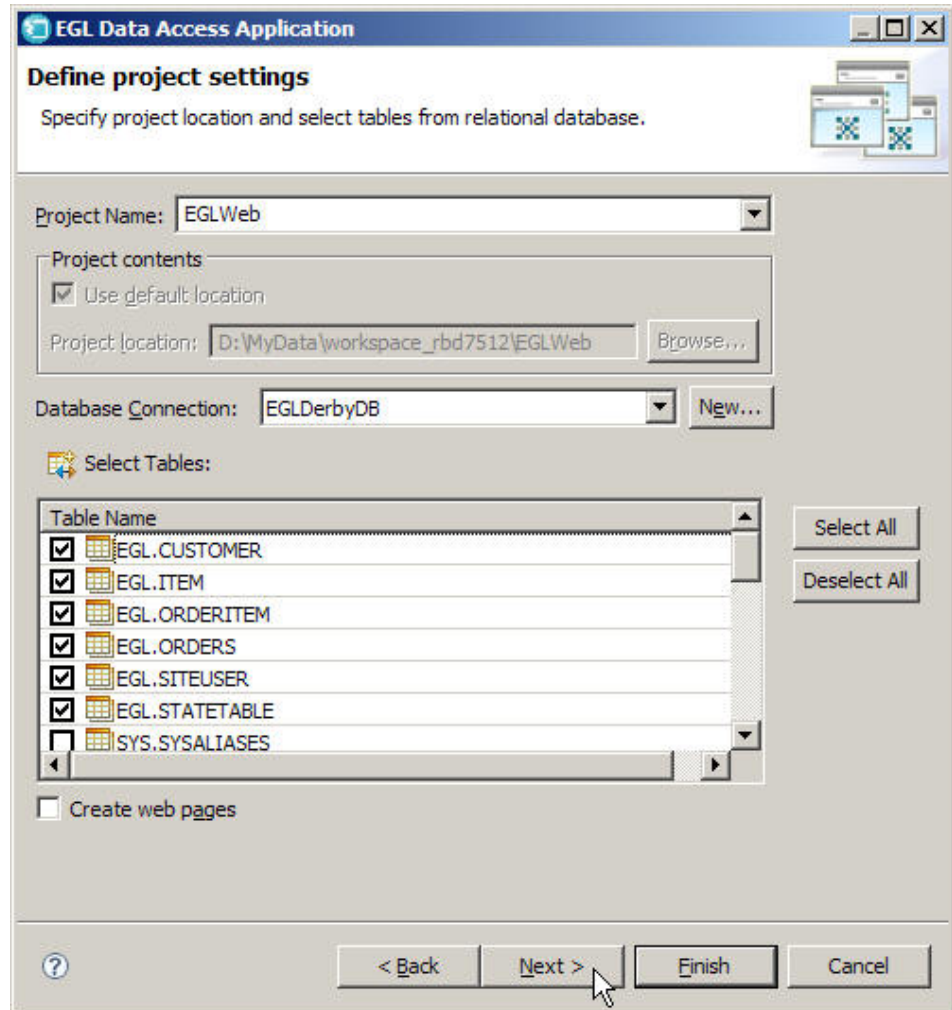
Create parts from the database connection

In the previous lesson, you created a connection to the database. From that connection, EGL can create the necessary parts to access the database:

Show Me

1. Click **File > New > Other**. The New window opens.
2. Expand **EGL** and click **Data Access Application**.
3. Click **Next**.
4. In the Define project settings window, make sure **EGLWeb** is displayed in the **Project Name** list.
5. In the **Database Connection** list, select the EGLDerbyDB connection that you created in the previous lesson. You might be prompted for a User ID and password for the database. If so, enter any name and password; the database does not require a password, but the workbench data tools might expect a password. You have established a connection to the database. All of the tables in the database are listed under **Table Name** at the bottom of the wizard. You will not create data parts for all of these tables because some contain only metadata.
6. Under **Select Tables**, select the check boxes next to the following tables only:
 - EGL.CUSTOMER
 - EGL.ITEM
 - EGL.ORDERITEM
 - EGL.ORDERS
 - EGL.SITEUSER
 - EGL.STATETABLE

The EGL Data Access Application wizard looks like this:



7. Make sure the **Create web pages** check box is clear. If this check box is selected, EGL creates web pages directly from the database tables. This feature can save time by creating a simple web data access application for you, but that would defeat the purpose of this tutorial.

Note: Do not click Finish yet. You must change one more setting in this wizard.

8. Click **Next** to move to the Define the Fields page. This page lets you add key fields to the database tables. Don't change any settings on this page; the database already has a key field in each table.
9. Click **Next** again to move to the Define project creation options page.
10. On the Define project creation options page, select the **Qualify table names with schema** check box.
11. Click **Finish**.

Lesson checkpoint

The Data Access Application wizard has created several EGL artifacts in your EGLWeb project.

First of all, there are several new EGL packages in the EGLSource folder of your EGLWeb project, including `eglderbydb.data`, `eglderbydb.access`, and

`eglderbydb.primitivetypes.data`. Packages work just like folders: they contain your source code files and organize them into meaningful groups. In this case, the `eglderbydb.data` package holds the records, the `eglderbydb.access` package holds the libraries, and the `eglderbydb.primitivetypes.data` package holds the `DataItems`.

Here are some of the files that will be useful for this tutorial:

`eglderbydb.primitivetypes.data.DataDefinitions.egl`

This file lists all of the `DataItems` that make up the Record parts in the other files. For example, the customer ID number given to each customer record in the database is represented by a `DataItem` named `CUSTOMER_ID`:

```
dataitem CustomerId INT end
```

In this case, the ID number field is based on the integer primitive type. The `DataItem` can have other properties to specify details like its valid range of values and how it should be formatted in the UI.

`eglderbydb.data.Customer.egl`

This file contains one of the records created from the database tables, in this case the Customer table. This record contains fields to hold information about a customer, such as the customer's first name, last name, address, telephone number, and ID number. The record definition looks like this:

```
record Customer type sqlRecord {
    tablenames=[["EGL.CUSTOMER"]],
    fieldsMatchColumns = yes,
    keyItems=[CUSTOMERID]
}

CUSTOMERID CUSTOMERID {column="CUSTOMERID"};
FIRSTNAME FIRSTNAME {column="FIRSTNAME", sqlVariableLen=yes, maxlen=30, isSqlNullable=yes};
LASTNAME LASTNAME {column="LASTNAME", sqlVariableLen=yes, maxlen=30, isSqlNullable=yes};
...
end
```

`eglderbydb.access.CustomerLib.egl`

This file contains an automatically-generated library of functions that you can use to access the Customers table of the database. For example, the first function is `AddCustomer`, which adds a new customer record to the database. There are other functions in this library that retrieve, update, and delete records in the database, and each table has similar functions in a separate library. You will use these functions later in the tutorial.

Lesson 5: Create a web page

In this lesson, you will create a web page in the form of a Faces JSP file. In the next lesson, you will add data to this page using the data parts and functions you created in the previous lesson. When the page is finished, it will show a list of every record in the database.

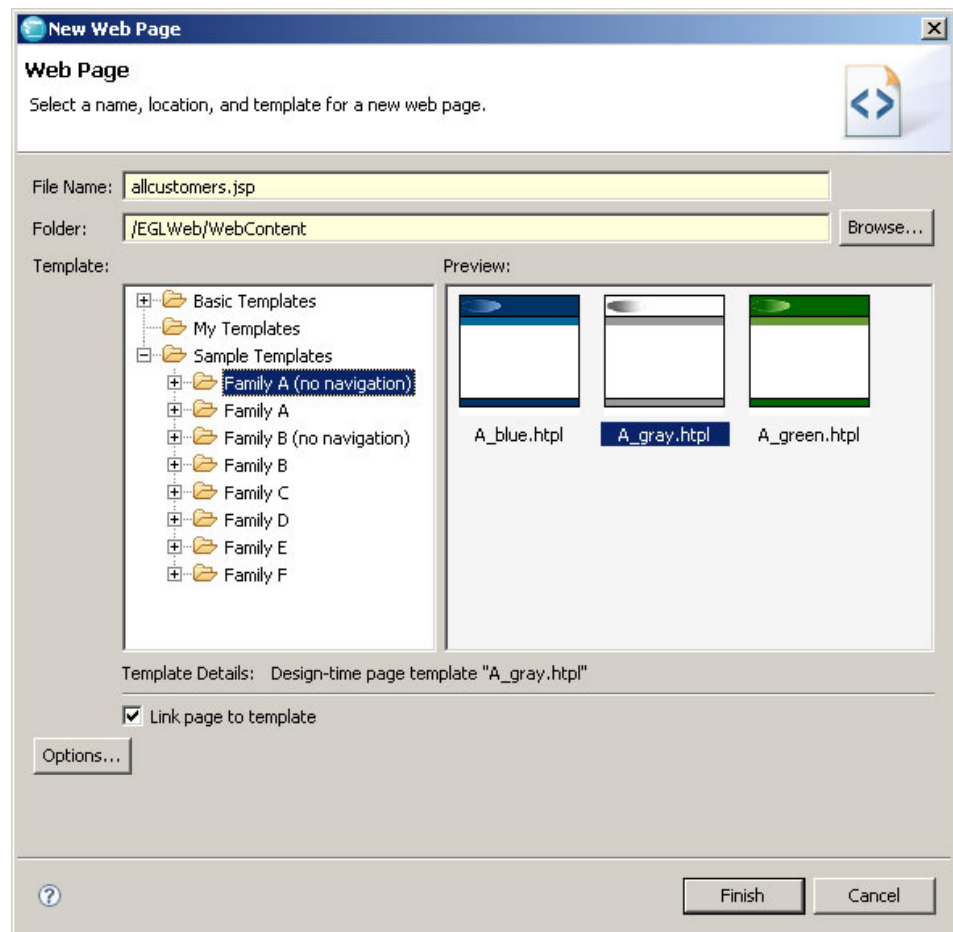
This tutorial uses JavaServer Faces (JSF) technology. JSF offers a framework for developing user interfaces for web applications. The web pages in JSF are JavaServer pages (JSPs). JSPs contain JavaServer controls that you can use to embed Java code into the page, providing dynamic content.

Create the JSP file from a template

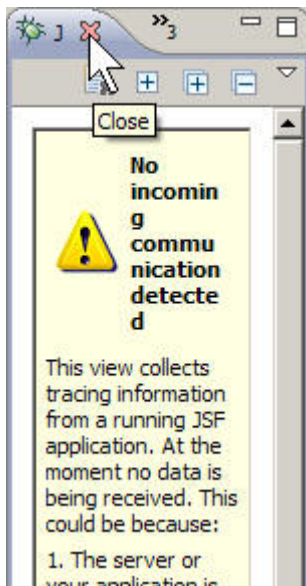
Show Me

1. In the Enterprise Explorer view, right-click the **WebContent** folder in the EGLWeb project, then click **New > Web page**. It is important to select the place where you want to put new files; otherwise the new files might not appear in the place you expect them to. The New Web Page window opens.
2. In the **File Name** field, type this file name, including the extension:
allcustomers.jsp
3. Make sure that the **Folder** field lists the /EGLWeb/WebContent folder.
4. In the **Template** list, expand **Sample Templates** and click **Family A (no navigation)**. The simple web page templates in this category are shown in the **Preview** box.
5. In the **Preview** box, click the **A_gray.html** template.
6. Make sure the **Link page to template** check box is selected.

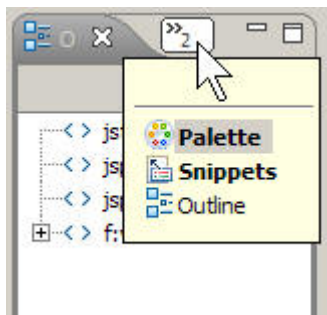
The New Web Page window looks like this:



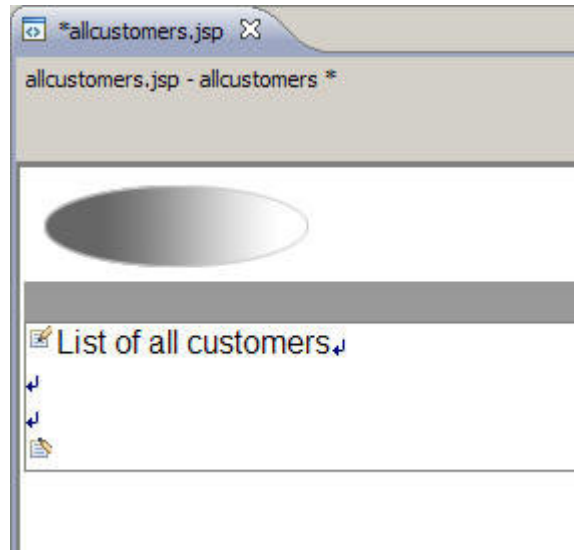
7. Click **Finish**. The new Faces JSP file opens in the editor.
8. Click **Design** to display the design view in the full editor window. You might see the JSF Trace view to the right of the editor window. Eclipse creates this view by default, sharing space with the Palette view and others. You do not need the JSF Trace view for this tutorial. You can close it by clicking the X in the tab.



Typically, you want to display the Palette in this view. You might need to click the small double arrow and select the Palette view from a menu:



9. In the new `allcustomers.jsp` file, remove the default text that says "Place your page content here" and replace it with the following text:
List of all customers
10. Press **Enter** three times to insert blank lines. These lines leave room for you to add content to this page in the next lesson.
The page looks like this:

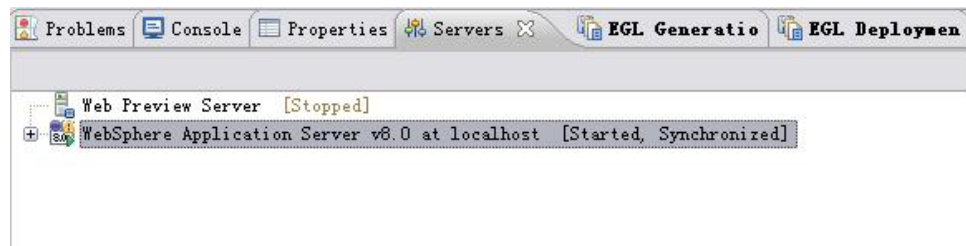


11. Save the file.

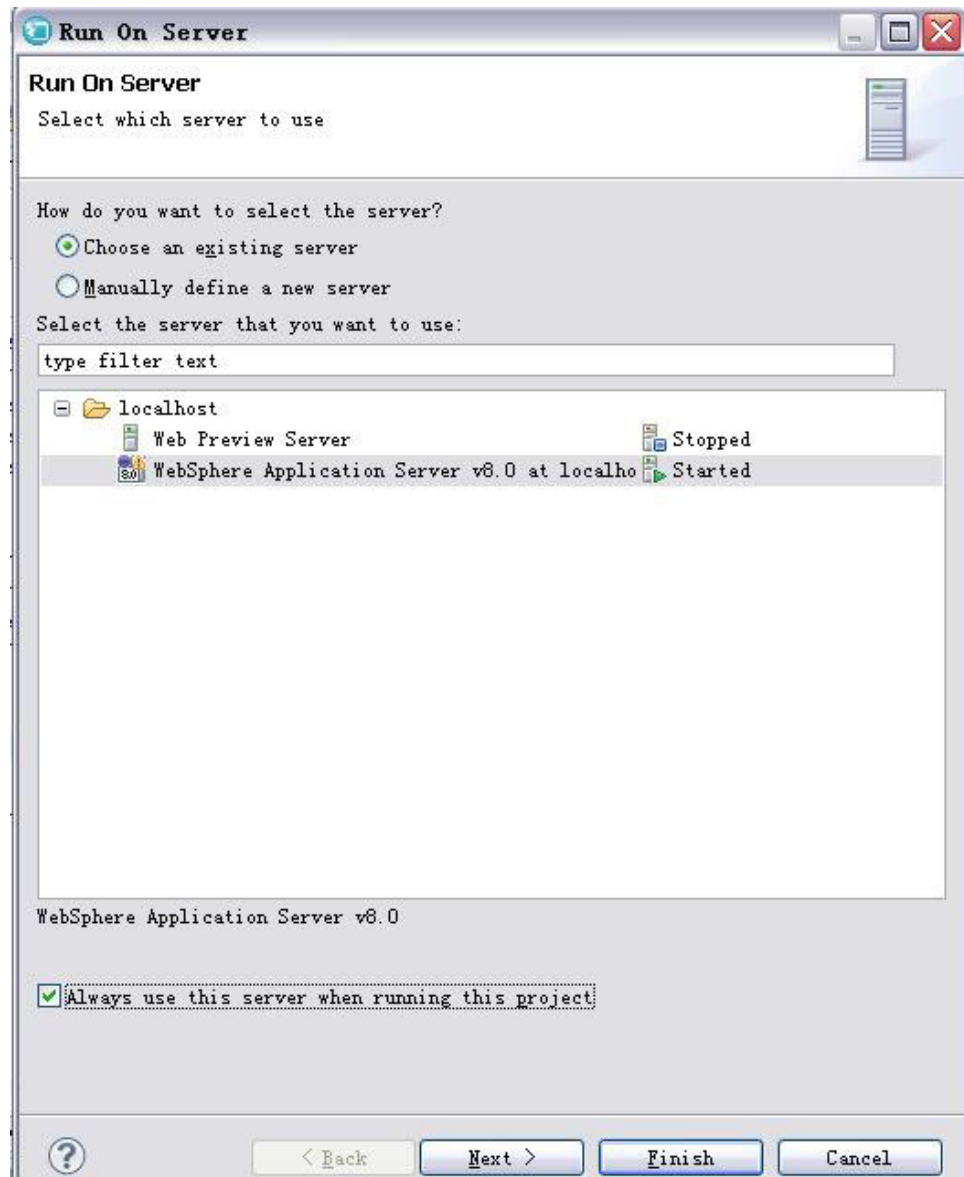
Preview the web page on the server

No data from the database is on the web page yet, but you can run the JSP file on the web application server and see how the page looks so far. This is an important step; it ensures you have the application server working properly before the page becomes complicated.

1. Make sure your application server is started. The server name should be visible in the **Servers** view (located by default under the Editor view). The server should show the words **Started** and **Synchronized** in brackets after the name. If not, right-click the server name and click **Start**. The start process might take a few minutes.

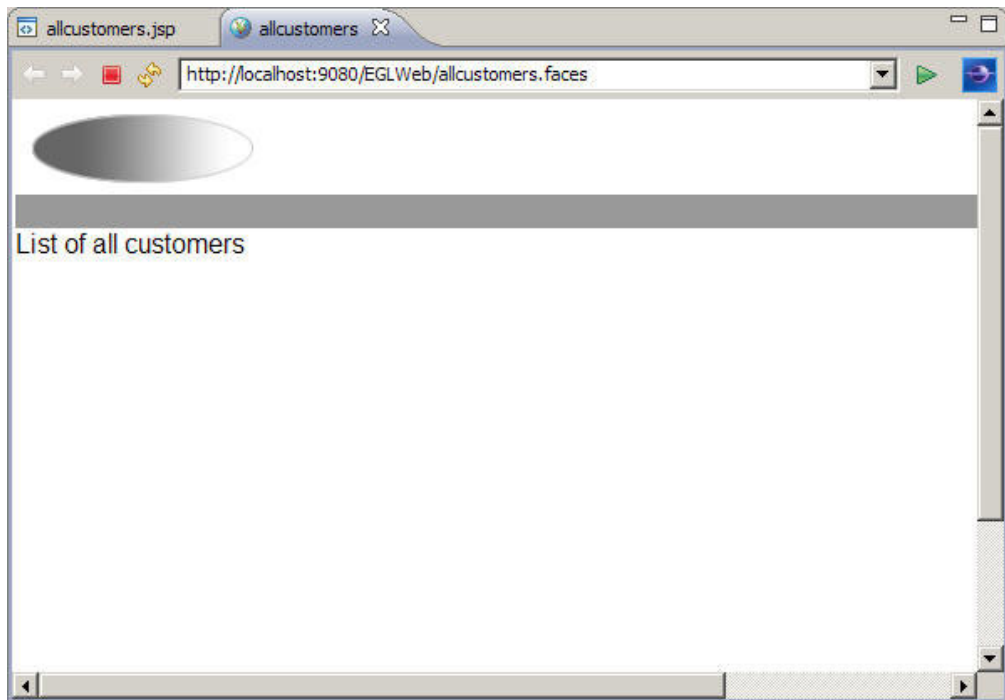


2. In the Enterprise Explorer view, expand the EGLWeb/WebContent folder if necessary. Right-click the **allcustomers.jsp** file and click **Run As > Run on Server**. The Define a new server window opens.
3. In the Define a New Server window, select your server.
4. Select the **Always use this server when running this project** check box.



5. Click **Finish**.

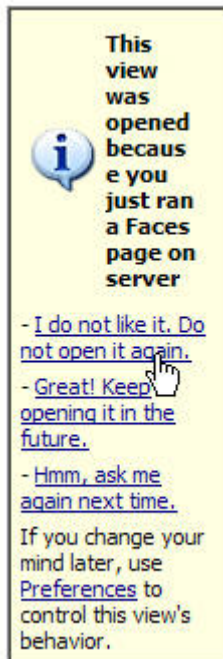
The web page opens in a web browser inside the workbench. The page looks like this:



If you see an HTTP 404 error message (page not found), restart the server and refresh the browser window.

If you prefer to use an external web browser, you can copy the URL from the web browser inside the workbench and paste that URL into the external browser's address field.

Eclipse may again display the JSF Trace view. You can permanently remove the view by scrolling down until you find a link that says **Do not open it again** and clicking that link.



This page does not have any data on it yet. In the next lesson, you will add data to this page using EGL.

Lesson 6: Add data to the page

In this exercise, you will add data from the database included with this tutorial onto the web page that you created in the previous exercise.

This task has the following parts:

- You add fields to the Page Data view, associating those fields with the web page.
- You place fields from the Page Data view onto the related web page.
- You add a variable (in this case, an array of records) to an EGL JSF handler, which you can think of as the code that stands behind the runtime process. The JSF handler is an example of page code because it can do any of these tasks:
 - Assign data values for submission to the JSP file. Those values are ultimately displayed on the web page.
 - Manipulate the data returned from the user or from a called program.
 - Forward control to another JSP file.

In this lesson you will also use the EGL content assist feature, a tool that you can use to complete programming statements without having to type the entire statement.

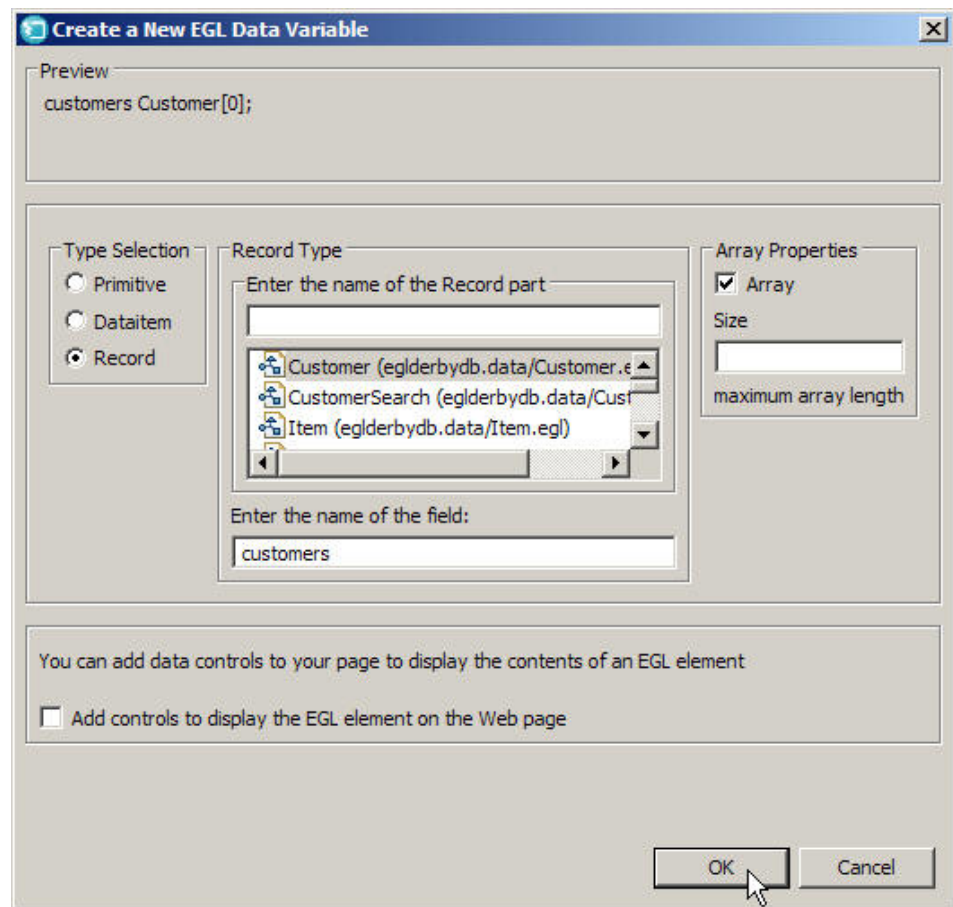
Add a record array to the Page Data view and the JSF Handler

Show Me

1. If the **allcustomers.jsp** file is not open, open it by double-clicking it in the Enterprise Explorer view.

2. Find the Page Data view, which is usually at the bottom left of the workbench. You can reveal the Page Data view by using the tabs, but if you can't find it, click **Window > Show View > Page Data**.
3. Find the Palette view, which is usually at the right side of the workbench. If you cannot find that view, click **Window > Show View > Basic > Palette**.
4. In the **Palette** view, click the **EGL** drawer to open it.
5. Drag the **New Variable** icon from the Palette view to the allcustomers.jsp page in the editor. The Create a New EGL Data Variable window opens.
6. Under **Type Selection**, click **Record**.
7. Under **Record Type**, click **Customer**. In this way, you select the Record part on which each of the array elements will be based.
8. In the **Enter the name of the field** field, type this text:
customers
9. Under **Array Properties**, select the **Array** check box. Leave the **Size** field blank.
10. Clear the **Add controls to display the EGL element on the web page** check box.

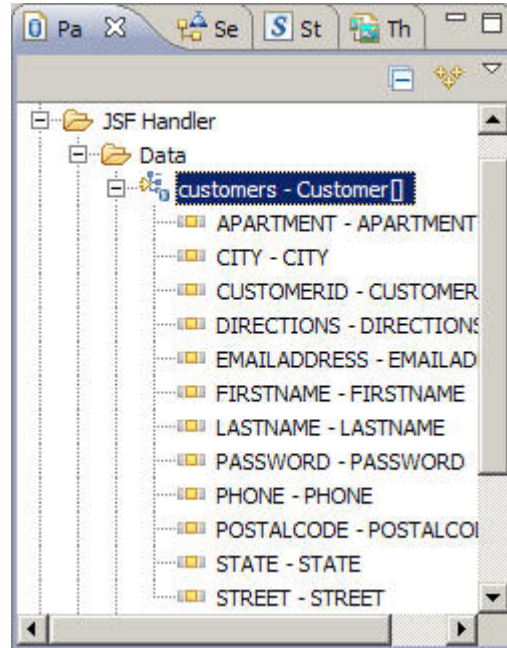
The Create a New EGL Data Variable window looks like this:



11. Click **OK**. Nothing appears on the JSP where you dragged the variable. The reason is that EGL created the variable in a separate file, called the JSF Handler, which contains code to respond to events in the JSP. An item representing the new variable appears in the Page Data view under **JSF Handler > Data**.

12. Expand **JSF Handler > Data** and then expand **customers - Customer[]**. There are 12 icons beneath customers, representing the 12 fields in this database record.

The Page Data view looks like this:



By adding entries to the Page Data view, you have also added an array of records to the JSF handler. In the next section, you will create the related fields on the web page.

Display the data on the web page

Data that is listed in the Page Data view can be added to the web page.

1. From the Page Data view, drag the **customers - Customer[]** array variable onto the file `allcustomers.jsp`, releasing it below the List of All Customers text, in the blank lines you added in the previous exercise.

The Insert List Control window opens. This window lists all of the fields in the database record. You can use this window to choose which fields to display on the page.

2. Under **Data control to create**, leave the default value, a **Multi-Column Data Table**.

3. Under **Create controls for**, click **Displaying an existing record (read-only)**.

With this option selected, the data on the page is displayed in read-only output fields. If you choose **Updating an existing record**, the fields on the page are input fields that you are able to type into, and beneath the fields will be buttons for you to bind actions to. You'll create this type of field on another page. For our purposes, the **Creating a new record** option is the same as **Updating an existing record** except that the default buttons are different.

4. Under **Columns to display**, click the **None** button. You have deselected all of the fields.
5. Select the check boxes next to these fields:

- CUSTOMERID
- FIRSTNAME
- LASTNAME

The Insert List Control window looks like this:

Insert List Control

Configure Data Controls
Specify the columns to display and how to display them

Data control to create: Multi-Column Data Table (one table row per data entry)

Create controls for:

☒ Displaying an existing record (read-only)

☐ Updating an existing record

☐ Creating a new record

Columns to display:

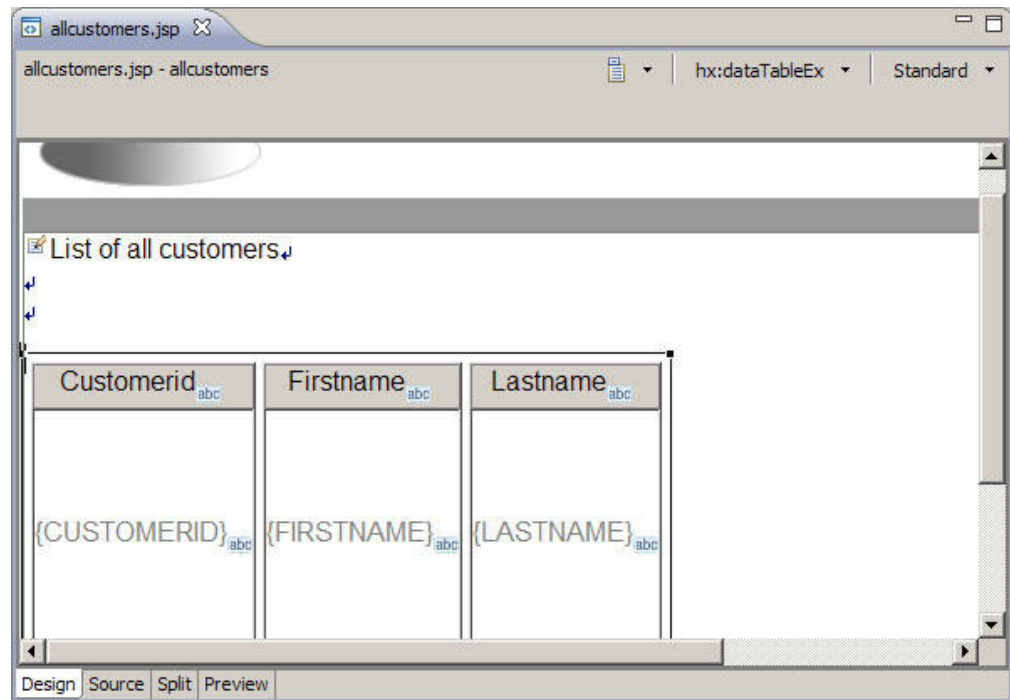
Column Name	Label	Control Type
<input checked="" type="checkbox"/> CUSTOMERID (int)	Customerid	Output field
<input checked="" type="checkbox"/> FIRSTNAME (string)	Firstname	Output field
<input checked="" type="checkbox"/> LASTNAME (string)	Lastname	Output field
<input type="checkbox"/> PASSWORD (string)	Password	Output field
<input type="checkbox"/> PHONE (string)	Phone	Output field
<input type="checkbox"/> EMAILADDRESS (string)	Emailaddress	Output field

All None Options... [Configure Control Types](#)

Finish Cancel

6. Click **Finish**. A data table is created on your page with three columns for the three fields you selected in the Insert List Control window.
7. Save the page.

The page looks like this example:



The columns in the data table have headings based on the names of the fields in the database. You can change these headings by clicking them, opening the Properties view, and changing the **Value** field.

The three text fields in the data table, which appear as {CUSTOMERID}, {FIRSTNAME}, and {LASTNAME}, represent the places where the database information will appear on the page.

Call a function from the EGL library

The next step is to add code to this page that calls a function in the CustomerLibrary.egl library. That function reads the data from the database and makes it available to the page.

Show Me

1. Right-click anywhere in the free-form area of the file allcustomers.jsp.
2. From the popup menu, click **Edit Page Code**.

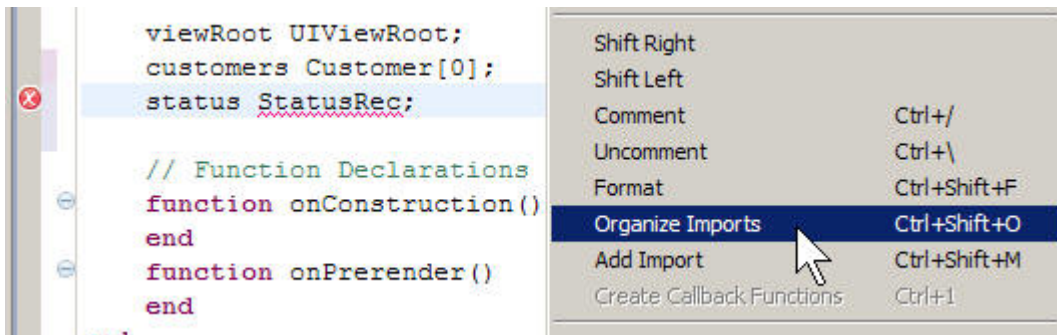
The allcustomers.egl file opens in the editor. This file holds a JSF Handler part. In the next steps, you add code to this JSF Handler that retrieves data from the database and puts it on the page.

3. In the allcustomers.egl file, find the line `customers Customer[0];`
This is the line of code that defines the record variable you created to display on the page. You also need to define a record to store the success or failure code of the SQL call.
4. On a blank line immediately after the line `customers Customer[0];`, add the following code, exactly as written:

```
status StatusRec;
```

Notice the wavy red line under StatusRec, indicating that the type is not known to the handler. You need to add an **import** statement that tells the handler where to find the StatusRec record definition. You can do this

automatically by right-clicking the blank space in the editor window and clicking **Organize Imports**.



The keyboard shortcut for this feature is Ctrl+Shift+O. EGL checks all files visible to it, locates the necessary information, and adds an **import** statement at the top of the file.

Now you have the record to be retrieved from the database and the SQL status record. The final step in adding the data to your page is to pass these two variables to the function that accesses the database. This function, named `GetCustomerListAll()`, was created by the Data Access Application wizard in a previous lesson.

Note the lines within braces that follow the **handler** declaration. These lines assign values to *properties* of the JSF Handler. In EGL, properties are name-value pairs that modify how a part behaves. Most types of EGL parts have one or more properties, and each kind of part can have different properties. In this case, the JSF Handler has four properties defined:

onConstructionFunction = onConstruction

The **onConstructionFunction** property specifies a function in the JSF Handler that runs the *first* time the web page (JSP) associated with the JSF Handler is displayed in a browser. In this case, the property specifies a function named `onConstruction`, which is created by default in the JSF Handler. You will not be working with this function in this tutorial.

onPreRenderFunction = onPreRender

The **onPreRenderFunction** property specifies a function in the JSF Handler that runs *each* time the associated JSP is displayed in the browser, including when the user refreshes the page or returns to the page after viewing another page. In this case, the property specifies a function named `onConstruction`, which is created by default in the JSF Handler. In the next few steps, you'll add code to this function to retrieve current data from the database each time the page loads.

view = "allcustomers.jsp"

The **view** property specifies the web page associated with the JSF Handler. By default, the web page and the JSF Handler have the same name, minus the file extensions.

viewRootVar = viewRoot

You use the `viewRoot` variable to get access to the JSF component tree. You will not use the `viewRoot` variable in this tutorial. For more about JSF components and the `viewRoot` variable, see Component tree access.

5. Add the code to call the `GetCustomerListAll()` library function to the `onPreRender()` function. This function retrieves the customer data from the database. In this case, try using the content assist tool in the EGL editor:
 - a. Place the cursor on a blank line between function `onPreRender()` and `end`.
 - b. Type the following code:


```
cust
```
 - c. Press **Ctrl+Spacebar**. The code completion window opens with all of the available EGL commands and resources beginning with `cust`.
 - d. From the content assist window, select the `CustomerLib` library either by highlighting it with the keyboard and pressing **Enter** or by double-clicking it with the mouse.

Now the new line of code reads `CustomerLib`.
 - e. Type a period after `CustomerLib`.
 - f. Press **Ctrl+Spacebar** again. The code completion window opens again.
 - g. From the code completion window, select the `GetCustomerListAll(customerArray Customer[], status StatusRec)` function either by highlighting it and pressing **Enter** or by double-clicking it with the mouse. Be careful not to select the function `GetCustomerList(listSpec ListSpecification, listOut Customer[], status StatusRec)`.

Now the new line of code reads `CustomerLib.GetCustomerListAll(customerArray, status)` and the `customerArray` argument is highlighted.
 - h. Change the default `customerArray` argument in the new line of code to the name of your record variable: `customers`.
 - i. End the line of code with a semicolon.

Finally, the new line of code reads:

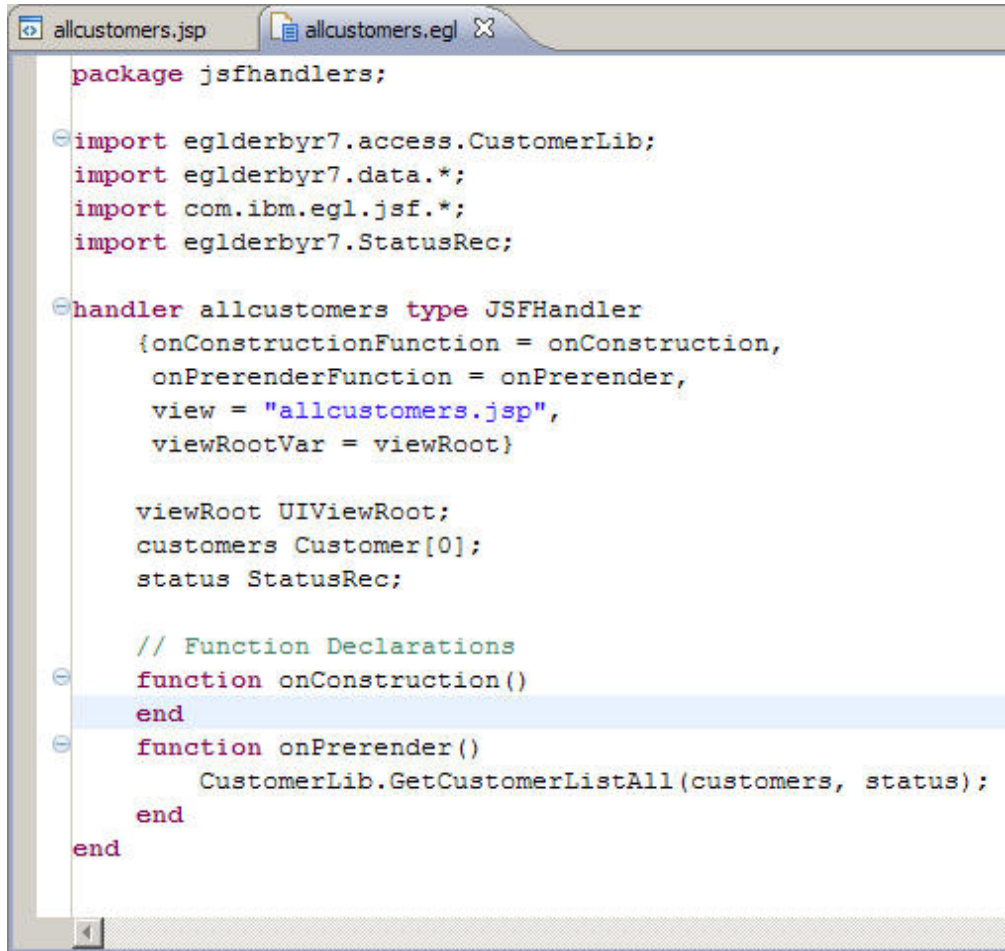
```
CustomerLib.GetCustomerListAll(customers, status);
```

Also note that there is a new import statement near the top of the file that reads `import eglderbydb.access.CustomerLib;` This line imports the library so you do not need to write out the complete path to the library in your code and instead can refer to it directly.

The content assist added this import statement automatically. If you had not used the content assist or the Organize Imports feature to create this import statement, you would have to specify the explicit location of the library, qualifying the library name with the following names: `eglderbydb.access.CustomerLib`.

6. Save the file.

The `allcustomers.egl` file now looks like the following example:



```
package jsfhandlers;

import eglderbyr7.access.CustomerLib;
import eglderbyr7.data.*;
import com.ibm.egl.jsf.*;
import eglderbyr7.StatusRec;

handler allcustomers type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
 view = "allcustomers.jsp",
 viewRootVar = viewRoot}

viewRoot UIViewRoot;
customers Customer[0];
status StatusRec;

// Function Declarations
function onConstruction()
end
function onPrerender()
    CustomerLib.GetCustomerListAll(customers, status);
end
end
```

If you see any errors marked by red X symbols in the editor, make sure your code matches the code in this file: “Completed allcustomers.egl file after lesson 6” on page 51.

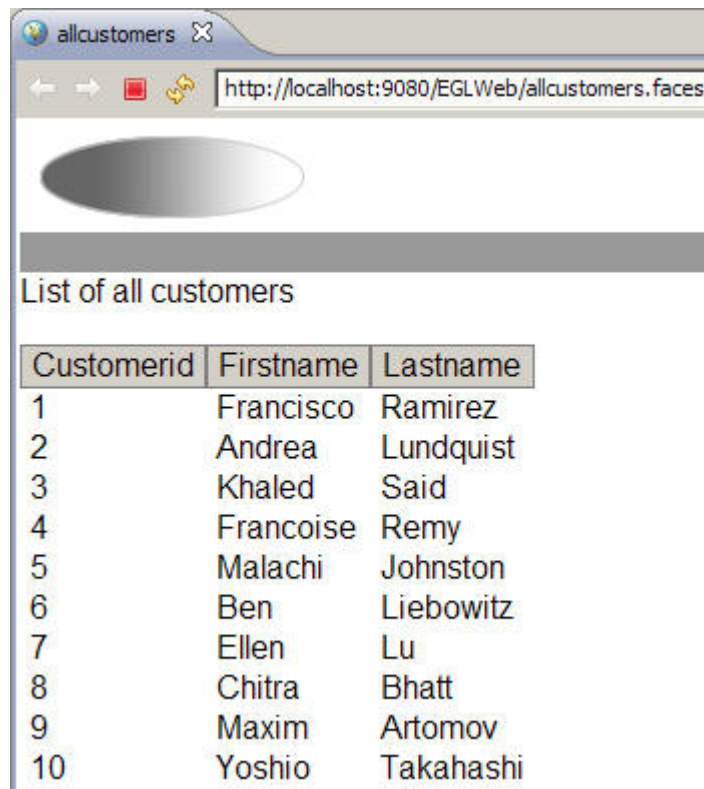
Test the page

Now the page is ready to be run on the server. Follow these steps to test it and see how the database data appears on the page.

Before proceeding, take the following precautions:

1. Save both the allcustomers.egl and allcustomers.jsp files if you have not already done so, then close both files.
2. Make sure your application server is started and synchronized.
1. In the Enterprise Explorer view, right-click the **EGLWeb** project and then click **Generate**.
2. In the Enterprise Explorer view, right-click the **allcustomers.jsp** file, not the allcustomers.egl file.
3. From the popup menu, click **Run As > Run on Server**.

As in the previous lesson, the web page opens in a web browser inside the workbench. This time, the dynamic data appears on the page. If you do not see the dynamic data, click the refresh icon next to the address bar. The page looks like this:



Customerid	Firstname	Lastname
1	Francisco	Ramirez
2	Andrea	Lundquist
3	Khaled	Said
4	Francoise	Remy
5	Malachi	Johnston
6	Ben	Liebowitz
7	Ellen	Lu
8	Chitra	Bhatt
9	Maxim	Artomov
10	Yoshio	Takahashi

In the next lesson, you will create a detail page to show all the fields in an individual customer record.

Lesson 7: Pass a parameter to another page

The file `allcustomers.jsp` lists every row in the database. In the next exercise, you will create a second page that displays the details from one row in the database. In this exercise, you will add a link on the file `allcustomers.jsp` that sends the user to the detail page. That link also indicates which record to display on the detail page.

Add the link to `allcustomers.jsp`

Show Me

1. Open the `allcustomers.jsp` file
2. In the Palette view, click the **Enhanced Faces Components** drawer to open it.
3. From the **Enhanced Faces Components** drawer, click the **Link** control to select it, not the Link - Request control.
4. With the **Link** control selected in the Palette view, click directly on the `{LASTNAME}` text control. Do *not* drag the Link control onto the JSP. The Configure URL window opens.
5. In the **URL** field of the Configure URL window, type the following file name exactly as shown:

`updatecustomer.faces`

This is the name of the page you will create in the next lesson to show only one row in the database, but with a `faces` extension instead of a `jsp` extension. The `faces` extension tells EGL that the file is to be treated as a JavaServer Faces file.

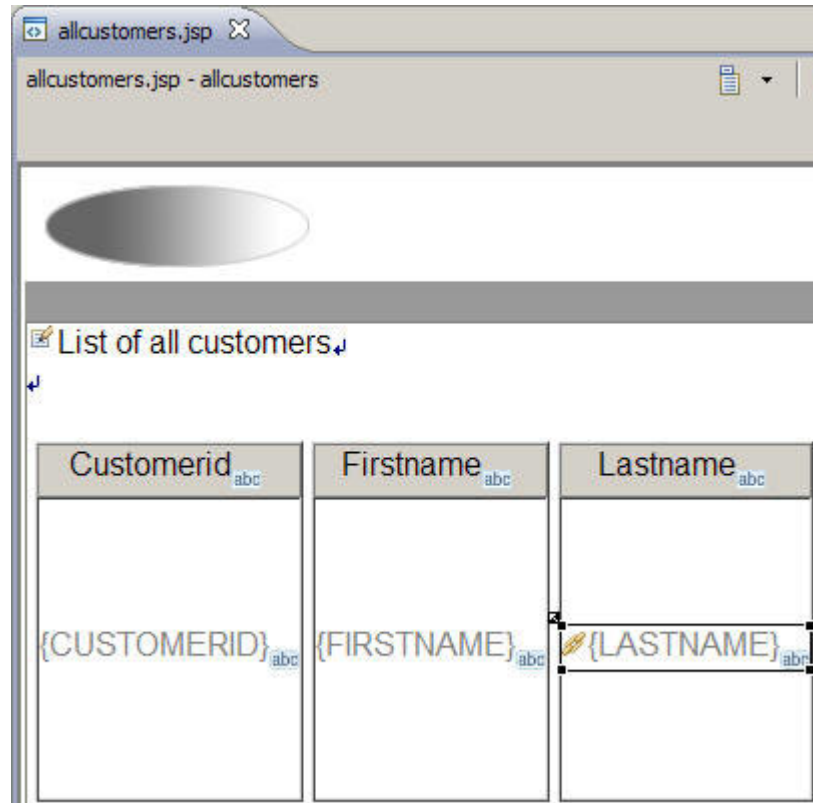
Leave the **Label** field blank. Without a label specified here, the link will use the text of the last name field itself as the text for the link.

6. Click **OK**.

If you see a link next to the {lastName} control named **Link label**, you did not place the link directly onto the {lastName} control. Click **Edit > Undo** and try again. When the link is placed correctly, you see a chain link icon next to the {LASTNAME} text control.

7. Save the page.


The page looks like this:



Add the parameter to the link

Next, you must specify which record will be displayed on the updatecustomer.jsp page. To send this information to that page, you specify an HTTP request parameter for the link you just added. HTTP request parameters are name-value pairs of plain text that are sent over the Internet by way of the HTTP protocol. Request parameters are an efficient way to send and receive simple data between programs within an application.

1. Click directly on the link icon of the link control you just added to the {LASTNAME} control.


The link icon itself, , not the text control, must be selected before you can continue. You have the link selected correctly if it is lightly shaded and the selection box is surrounding the link icon and the text control. Do not double click the link icon.

2. Without moving the selection away from the link icon, open the Properties view.

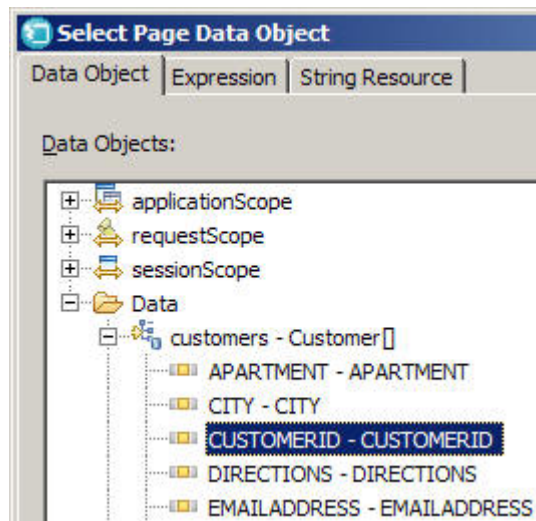
The Properties view is usually at the bottom of the workbench. If you can't find the Properties view, click **Window > Show View > Properties**.

3. In the Properties view, click the **Parameter** tab, directly below the **hx:outputLinkEx** tab.

If you can't find the **Parameter** tab, be sure you have clicked directly on the icon to select it.

4. Click **Add Parameter**. A new parameter named Name0 is added to the list of parameters.
5. Click the cell holding Name0 and replace the name with the following text as the new name of the parameter:
CID
6. Click the cell holding Value0 to highlight it.
7. Click the  **Select Page Data Object** button. The Select Page Data Object window opens.
8. Under **Data Objects**, expand **Data**.
9. Expand **customers - Customer[]**.
10. Click **CUSTOMERID - CUSTOMERID**.

The Select Page Data Object window looks like this:



11. Click **OK**.
12. Save and close the page.

Now, the value of the CID parameter for the link is bound to the value of the customer_id field. When the user clicks the link, the runtime code invokes the file updatecustomer.jsp and makes the customer ID number available to the onPreRender function of the related JSF handler.

In the next lesson, you will create the web page for the file updatecustomer.jsp, and later, you will set up the JSF handler to receive the parameter and to show only the customer with that ID number.

Lesson 8: Create an update page

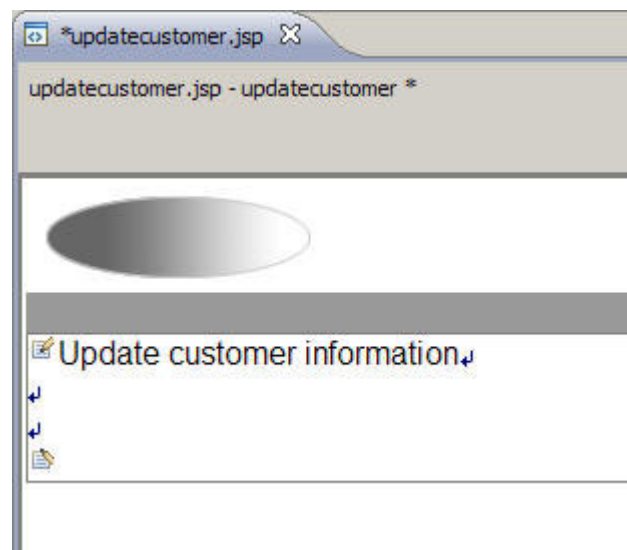
In this exercise, you will create the web page that allows users to update the CUSTOMER table. This page will receive the parameter that the other page passed, display only the record indicated by that parameter, and accept updated information for the record.

Create the updatecustomer.jsp file

Show Me

1. In the Enterprise Explorer view, right-click the **WebContent** folder of the EGLWeb project.
2. Click **New > Web page**.
3. In the **File Name** field, type this text as the name of the new file:
updatecustomer.jsp
4. Make sure that the **Folder** field lists the /EGLWeb/WebContent folder.
5. In the **Template** list, click **My Templates**.
6. In the **Preview** box, click the **A_gray.html** template.
7. Click **Finish**. The new page is created and opens in the editor.
8. Replace the default text with this text:
Update customer information
9. Press **Enter** three times to insert three blank lines.
10. Save the page.

The new updatecustomer.jsp page looks like this:



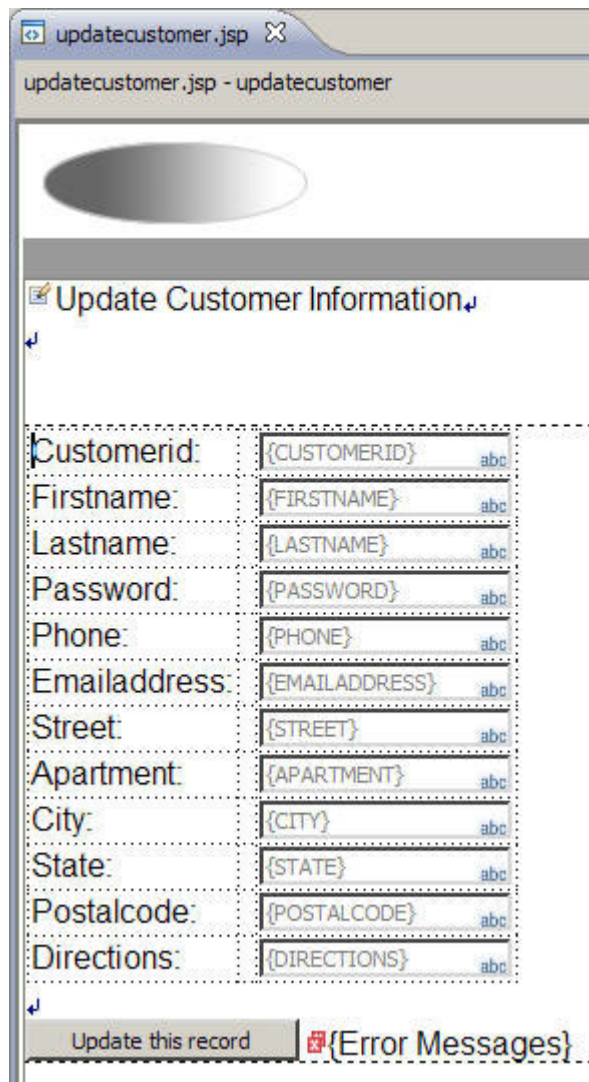
Add an EGL record and display it on the page

The next step is to add EGL data to this page. When you created the allcustomers.jsp file, you added the data to the page in one step and then displayed the data on the page by dragging it from the Page Data view in a second step. This time, you will select the **Add controls to display the EGL element on the web page** check box to add the data to the page and display it on the page in one step.

1. Open the **EGL** drawer on the Palette view.
2. Drag the **New Variable** icon onto the page, below the text "Update Customer Information." The Create a New EGL Data Variable window opens.
3. Under **Type Selection**, click **Record**.
4. Under **Record Type**, click **Customer**.
5. In the **Enter the name of the field** field, type the following text:
customer
6. Under **Array Properties**, clear the **Array** check box.
7. Select the **Add controls to display the EGL element on the web page** check box.
8. Click **OK**. The new record appears in the Page Data view and the Insert Control window opens.
9. In the Insert Control window, click **Updating an existing record**.
10. Click **Options**. The Options window opens.
11. Select the **Submit button** check box.
12. Clear the **Delete button** check box.
13. For the **Label** of the **Submit button**, type this text:
Update this record
14. Click **OK**.
15. Click **Finish**.
16. Save the page.

The data controls for updating the record are inserted on the web page. Note that there is an {Error Messages} control on the page. This control does not mean that your page has errors; the {Error Messages} control marks the place where run time error messages will be displayed.

The page looks like this:



Retrieve the data

Now that there are fields for the data on the page, you need to add the code that retrieves the data from the database. Recall from the previous lesson that you added a link to pass the customer ID number in a parameter named CID. In these steps, you will set up the handler for new web page to accept this parameter and retrieve the appropriate record from the database to be displayed on the page.

1. Right-click anywhere in the free-form area of the updatecustomer.jsp file.
2. From the popup menu, click **Edit Page Code**. The updatecustomer.egl file opens in the editor.
3. As in the previous JSF Handler you edited, you need to add a record to store the success or failure code of the SQL call. Immediately after the line `customer Customer;`, add the following code, exactly as written:

```
status StatusRec;
```

The next step in adding the data to the page is to configure the JSF handler to accept the CID parameter that the link will pass to it.

4. Change the line `function onPreRender()` to the following code, exactly as written:

```
function onPreRender(CID INT)
```

Now the JSF handler is configured to accept an integer parameter named CID.

5. On a blank line immediately after the function `onPreRender(CID INT)`, add this code, exactly as written:

```
customer.customerId = CID;
```

Now you have assigned the ID number to the customer record. The next step is to retrieve the record with this ID number from the database

6. On the next line, add this code, exactly as written. You may want to use the code completion feature you learned about in “Lesson 6: Add data to the page” on page 33.

```
CustomerLib.GetCustomer(customer, status);
```

The `GetCustomer` function works just like the `GetCustomerAll` function you used previously, but the `GetCustomer` function retrieves one record, while the `GetCustomerAll` function retrieves an array of records. Now the customer record contains the record with the ID passed to this JSF handler.

The new function looks like this:

```
function onPreRender(CID INT)
  customer.CustomerId = CID;
  CustomerLib.GetCustomer(customer, status);
end
```

7. Optimize imports and save the file.

The JSF handler looks like this:

```
package jsfhandlers;

import com.ibm.egl.jsf.UIViewRoot;
import eglderbydb.StatusRec;
import eglderbydb.access.CustomerLib;
import eglderbydb.data.Customer;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
 view = "updatecustomer.jsp",
 viewRootVar = viewRoot}

viewRoot UIViewRoot;
customer Customer;
status StatusRec;

// Function Declarations
function onConstruction()
end
function onPrerender(CID int)
    customer.customerId = CID;
    CustomerLib.GetCustomer(customer, status);
end
end
```

Now when you click a link on the allcustomers.jsp page, the updatecustomer.jsp page loads with details about that customer's record. Right now, you can change the information in the fields on the web page, but there is no function to send those updates to the database. In the next section, you will use the UpdateCustomer function to make those updates to the database.

Update the record in the database

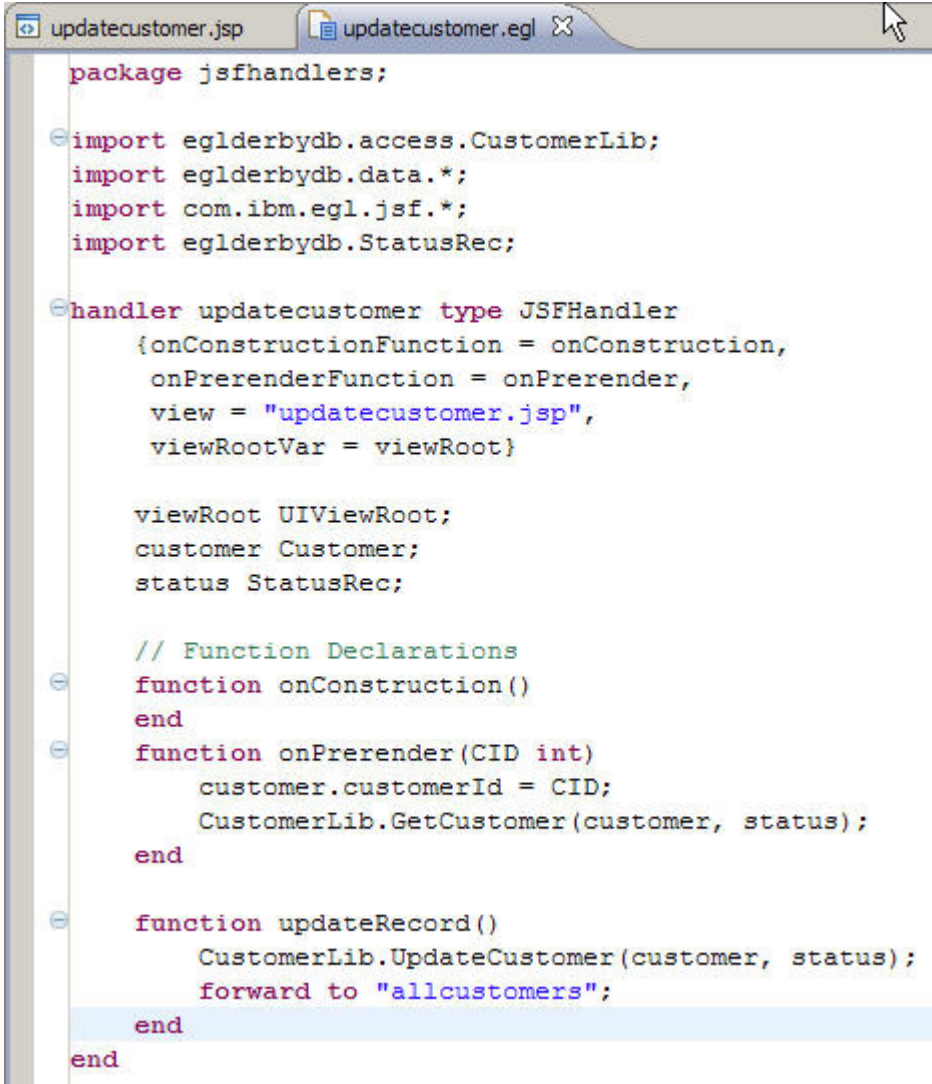
In this section, you add a new EGL function in the JSF handler named updateRecord. Then, you will bind this function to the button you created on the web page. In this way, when you click the button on the web page, the updateRecord function will run and call the UpdateCustomer function to update the database record. Finally, the updateRecord function will forward the browser back to the allcustomers.jsp page so you can see the changes you have made to the record.

1. In the updatecustomer.egl file, immediately before the final End statement, add the following function, exactly as shown. You might want to use code completion or to paste the function from this page to make sure it is correct.

```
function updateRecord()
    CustomerLib.UpdateCustomer(customer, status);
    forward to "allcustomers";
end
```

2. Save and close the file.
The next step is to bind this function to the button on the web page.
3. Open the updatecustomer.jsp page in the editor. You may still have this page open in the editor tabs. If you cannot find it there, double-click the updatecustomer.jsp file in the Project Explorer view, in the folder EGLWeb/WebContent.
4. In the Page Data view, expand **JSF Handler > Actions**. This folder lists all of the functions in the JSF handler except the onPreRender() and onConstruction() functions. In this case, this folder shows the updateRecord() function that you just created.
5. Drag the updateRecord() function directly onto the button on the web page labeled "Update this record". The appearance of the page does not change, but now this function is bound to the button and will run when the button is pressed.
6. Save the page.

Here is the complete code of the updatecustomer.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed updatecustomer.egl file after lesson 8" on page 52



```
package jsfhandlers;

import eglderbydb.access.CustomerLib;
import eglderbydb.data.*;
import com.ibm.egl.jsf.*;
import eglderbydb.StatusRec;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
 view = "updatecustomer.jsp",
 viewRootVar = viewRoot}

viewRoot UIViewRoot;
customer Customer;
status StatusRec;

// Function Declarations
function onConstruction()
end
function onPrerender(CID int)
    customer.customerId = CID;
    CustomerLib.GetCustomer(customer, status);
end
function updateRecord()
    CustomerLib.UpdateCustomer(customer, status);
    forward to "allcustomers";
end
end
```

Test the finished site

Now the site is ready to test. You can now update and view any of the records in the Customer table of the database.

1. In the Enterprise Explorer view, right-click the `allcustomers.jsp` file and then click **Run As > Run on Server**. The related page opens in the web browser. Now each customer last name in the list is a hyperlink to the web page displayed by `updatecustomer.jsp`.
2. Click one of the customer last names. You are sent to the web page displayed by `updatecustomer.jsp`, and that web page shows the row-specific information.
3. Type a new `FIRST_NAME` for the record.
4. Type new information for a few of the other fields on this page. Do not change the `CUSTOMER_ID` field.
5. When you are finished typing new information, click the **Update this record** button.

When you click the **Update this record** button, you return to the page `allcustomers.jsp`. Note that the record has changed to show the new `FIRST_NAME` you typed. You can click on the last name for that record again to see the new information that was saved in the database.

You have completed the tutorial

In this tutorial you built a functioning file maintenance utility for a customer file, using a sample Derby database. You can now build on this knowledge by completing the Build a JSF search page with EGL tutorial.

Summary

This is the end of the *Introducing EGL* tutorial.

Lessons learned

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

- Create EGL source code
- Create two simple web pages that access data in a relational database
- Pass a parameter from one web page to another
- Configure a web application server test environment and run an application on that test environment

You can continue learning by working with the tutorial application. Try adding this functionality on your own:

- Add a button to the `updatecustomer.jsp` page that deletes the customer record. You will need to add a button to the web page and then bind that button to a function in the JSF handler that calls the `deleteCustomer` function.
- Add a button to the `allcustomers.jsp` page that creates a new customer record. You will need to create a new web page similar to the `updatecustomer.jsp` page that uses the `createCustomer` function.

Resources

This tutorial used the following resources:

- A sample Derby database, found at the following location:

```
shared_resources/plugins/com.ibm.etools.egl.tutorial0001.doc_version/  
resources/eglderbydb.zip
```

shared_resources

The shared resources directory for your product, such as C:\Program Files\IBM\SDP70Shared on a Windows system or /opt/IBM/SDP70Shared on a Linux system. If you installed and kept a previous version of an IBM product containing EGL before installing your current product, you may need to specify the shared resources directory that was set up in the earlier installation.

version

The installed version of the plugin. If more than one is present, use the one with the most recent version number, unless you have a reason to use an older version.

- “Completed allcustomers.egl file after lesson 6”
- “Completed updatecustomer.egl file after lesson 8” on page 52

Following are some links to the help system on topics covered in this tutorial:

- Enabling EGL capabilities
- Introduction to EGL
- JavaServer Faces
- Scope
- EGL projects, packages, and files

Completed allcustomers.egl file after lesson 6

This code is the completed version of the allcustomers.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package jsfhandlers;  
  
import eglderbydb.access.CustomerLib;  
import eglderbydb.data.*;  
import com.ibm.egl.jsf*;  
import eglderbydb.StatusRec;  
  
handler allcustomers type JSFHandler  
{onConstructionFunction = onConstruction,  
  onPreRenderFunction = onPreRender,  
  view = "allcustomers.jsp",  
  viewRootVar = viewRoot}  
  
viewRoot UIViewRoot;  
customers Customer[0];  
status StatusRec;  
  
function onConstruction()  
end  
function onPreRender()  
  CustomerLib.GetCustomerListAll(customers, status);  
end  
end
```

Return to “Completed allcustomers.egl file after lesson 6.”

Completed updatecustomer.egl file after lesson 8

This code is the completed version of the updatecustomer.egl file. If you see any errors marked by red X symbols in the file, make sure your code matches this code:

```
package jsfhandlers;

import egl DerbyDB.access.CustomerLib;
import egl DerbyDB.data.*;
import com.ibm.egl.jsf.*;
import egl DerbyDB.StatusRec;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
 onPrerenderFunction = onPrerender,
  view = "updatecustomer.jsp",
  viewRootVar = viewRoot}

viewRoot UIViewRoot;
customer Customer;
status StatusRec;

// Function Declarations
function onConstruction()
end

function onPrerender(CID INT)
  customer.customerId = CID;
  CustomerLib.GetCustomer(customer, status);
end

function updateRecord()
  CustomerLib.UpdateCustomer(customer, status);
  forward to "allcustomers";
end
end
```

Return to “Lesson 8: Create an update page” on page 44.

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:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, 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
3600 Steeles Avenue East
Markham, ON Canada L3R 9Z7

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 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.

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 years_. All rights reserved.

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