



Introducing EGL (a quick-start guide)

Contents

Introducing EGL (a quick-start guide) . . . 1

Introduction 1

Lesson 1: Setting up EGL 5

 Preparing your workspace. 6

Lesson 2: Create the projects and import the database 6

 Installing the server (WebSphere Application

 Server) 7

 Installing the server (Tomcat). 8

 Create the EGL Web project 9

 Import the database 11

 Lesson checkpoint 13

Lesson 3: Set up the database connection 14

 Setting up the database connection (WebSphere

 Application Server). 15

 Setting up the database connection (Tomcat) . . 16

 Lesson checkpoint 17

Lesson 4: Create parts to access a database . . . 18

 Connect to the database and create parts . . . 18

 Lesson checkpoint 21

Lesson 5: Create a Web page 22

 Create the JSP file from a template 22

 Preview the Web page on the server 24

Lesson 6: Add data to the page. 25

 Add a record array to the Page Data view and

 the JSF Handler 25

 Display the data on the Web page 27

 Add a function call to the EGL library 29

 Test the page 32

Lesson 7: Pass a parameter to another page. . . . 32

 Add the link to allcustomers.jsp 32

 Add the parameter to the link 33

Lesson 8: Create an update page 34

 Create the file updatecustomer.jsp 34

 Add an EGL record and display it on the page 35

 Retrieve the data 36

 Update the record in the database 38

 Test the finished site 39

Summary 40

Resources 40

 Completed allcustomers.egl file after lesson 6 . . 41

 Completed updatecustomer.egl file after lesson 8 41

Introducing EGL (a quick-start guide)

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

Related information

Create a hello world service with EGL

Create a hello world program with EGL

Build a JSF search page with EGL



[View the PDF version](#)

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 have 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.

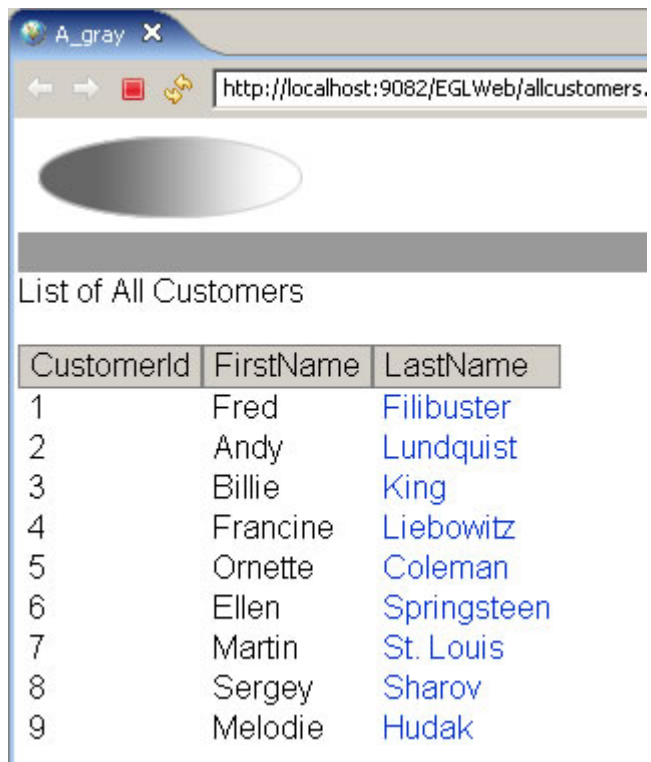
Prerequisites

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

A screenshot of a web browser window with the title 'A_gray'. The address bar shows 'http://localhost:9082/EGLWeb/updatecustomer'. The page content includes a large oval graphic and a form titled 'Update Customer Information'. The form has the following fields and values:

CustomerId:	1
FirstName:	Fred
LastName:	Filibuster
Password:	b1
Phone:	(201)652-3456
EmailAddress:	f_filiBust@uas.com
Street:	14 Maple Avenue
Apartment:	1A
City:	Wake Forest
State:	
Postalcode:	NJ
Directions:	07542

At the bottom of the form is a button labeled '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**.
4. Under **General**, click **Capabilities**.
5. Select the **EGL Developer** check box. If you do not see the EGL Developer folder in the right pane of the Preferences window under **Capabilities**, EGL is not installed on your system. Run the product setup again and select the **Additional Feature** item for EGL.

The **Capabilities** page looks like the following picture. You may have other capabilities available, depending on which products and options you have installed.



6. Click **OK**.

You have enabled the necessary EGL capability so that you can create EGL-related files and projects. There may be many other capabilities available on the capabilities page. You do not need to enable any other capabilities for this tutorial, but you must enable them if you decide to use those tasks.

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 or by clicking **Window** → **Open Perspective** → **Other** and clicking 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.

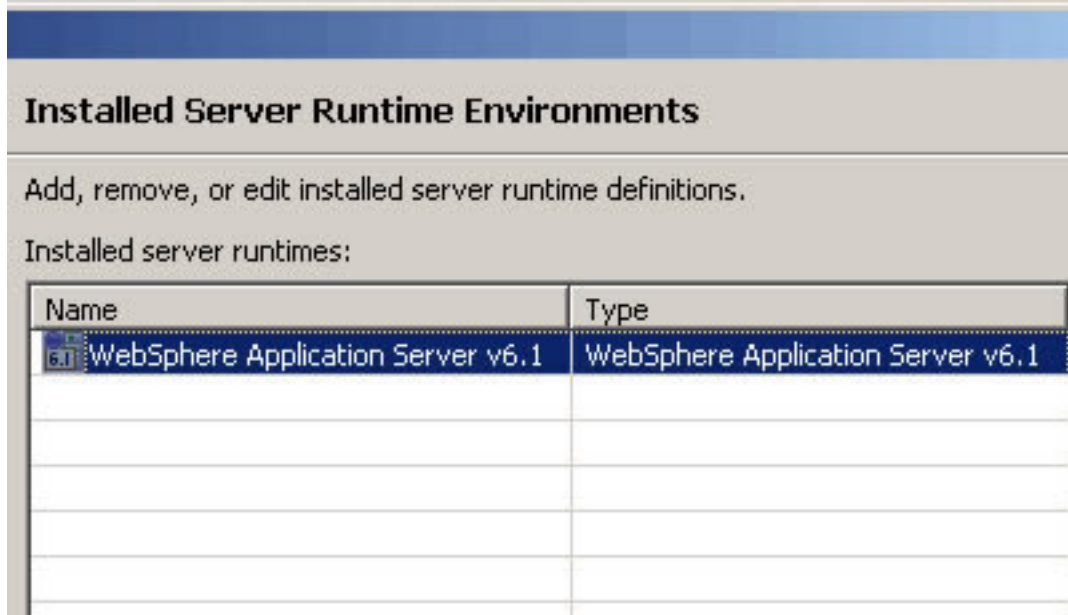
For the rest of the tutorial, be aware of sections that apply to WebSphere Application Server or to Apache Tomcat. If the instructions refer to projects, files, or options that you do not have, check to see that you are in the correct section for your server.

Installing the server (WebSphere Application Server)

If you are using WebSphere Application Server, it was probably installed along with EGL. Follow these steps to ensure that it is installed and configured:

1. From the menu bar, click **Window** → **Preferences**. The Preferences window opens.
2. At the left side of the Preferences window, expand **Server**.
3. Under **Server**, click **Installed Runtimes**. The right side of the Preferences window now shows a list of all of the server runtimes installed in the workbench.

- Under **Name**, look for an entry named **IBM® WebSphere Application Server, version 6.1**, as in this



picture:

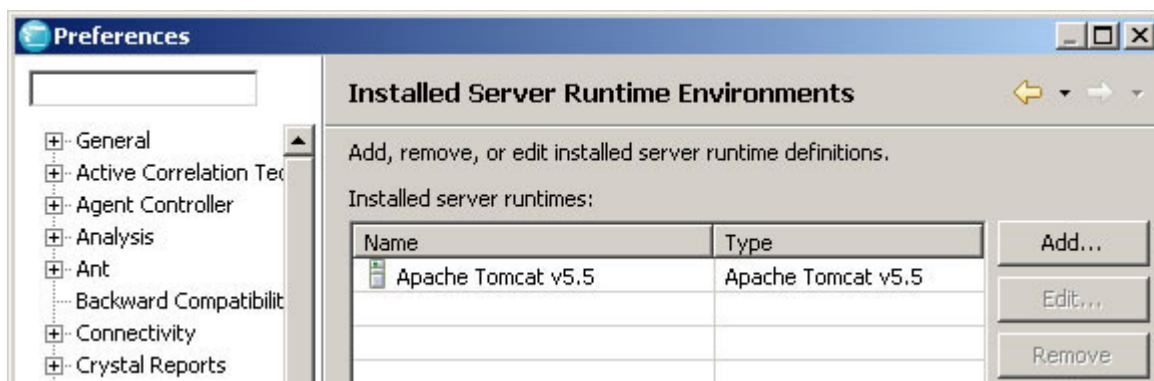
- If you see **IBM WebSphere Application Server, version 6.1** in the list, the test environment is installed. You may have other runtimes listed.
- If you do not see an entry named **IBM WebSphere Application Server, version 6.1**, you must run the product setup again. When you run the product setup, select the check box to install **IBM WebSphere Application Server, version 6.1**.
- Click **OK**.

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 5.5:

- In a Web browser, go to the Apache Tomcat Web page: <http://tomcat.apache.org>.
- Click the **Download** link for **Tomcat 5.x**.
- On the download page, find the most recent version of Tomcat with a version number like 5.5.x, such as 5.5.23. Several downloads are available for each version. Do not use versions marked as "alpha" or "beta."
- Under the 5.5.x version of Tomcat, look under **Binary Distributions** and then under **Core**.
- Download the **Core** version of Tomcat 5.5.x for your operating system. For Windows, select the ZIP file. For Linux, select the TAR.GZ file.
- Unzip the file you downloaded and place the server files in an appropriate location on your computer. It isn't important where you put the server files as long as you can find them again later.
- In the workbench, click **Window** → **Preferences**. The Preferences window opens.
- At the left side of the Preferences window, expand **Server**.
- Under **Server**, click **Installed Runtimes**. The right side of the Preferences window now shows a list of all of the server runtimes installed in the workbench.
- Click **Add**. The New Server Runtime window opens.
- In the New Server Runtime window, expand **Apache**.
- Under **Apache**, click **Tomcat v5.5**.
- Click **Next**.
- Under **Tomcat installation directory**, click **Browse**.
- Locate the folder that contains the Tomcat installation files and then click **OK**.

16. Click **Finish**. Tomcat is added to the list of installed runtimes. The Preferences window looks like this:



17. Click **OK**.

Create the EGL Web project

Because your project will have a Web page interface, you need to create an EGL Web project, not an EGL 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 **File** → **New** → **Project**. The New Project window opens.
2. Expand the **EGL** folder and click **EGL Web Project**, not EGL Project.
3. Click **Next**.
4. In the **Name** field, enter this name for your project:
EGLWeb
5. In the **JNDI name for SQL connection** field, enter this code, exactly as shown:
jdbc/EGLDerbyR7

This code tells the project where to find its data source. Later, you will associate a database with this JNDI name. Be sure to enter this name exactly as it is written; JNDI names are case-sensitive.

6. In the **Target Runtime** field, select your server:
 - If you are using WebSphere Application Server, select **WebSphere Application Server v6.1**.
 - If you are using Tomcat, select **Apache Tomcat v5.5**.
7. If you are using WebSphere Application Server, select the **Add project to an EAR** check box.
8. If you are using WebSphere Application Server, in the **EAR Project Name** field, type the following name for your EAR project:
EGLWebEAR

The New EGL Web Project window looks like this:

New Dynamic Web Project

New EGL Web Project
Create an EGL-enabled Web Project.

Project name:

Project contents:

☒ Use default

Directory:

Build Descriptor Options

☒ Create new project build descriptor(s) automatically

☐ Use build descriptor specified in EGL Preference

☐ Select existing build descriptor

JNDI name for SQL connection:

Target Runtime

Configurations

Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership

☒ Add project to an EAR

EAR Project Name:

9. If you are using Tomcat, the **Add project to an EAR** check box is disabled. The New EGL Web project window looks like this:

New Dynamic Web Project

New EGL Web Project

Create an EGL-enabled Web Project.

Project name:

Project contents:

☒ Use default

Directory:

Build Descriptor Options

☒ Create new project build descriptor(s) automatically

☐ Use build descriptor specified in EGL Preference

☐ Select existing build descriptor

JNDI name for SQL connection:

Target Runtime

Configurations

Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership

☐ Add project to an EAR

EAR Project Name:

10. Click **Finish**.
11. 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.

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

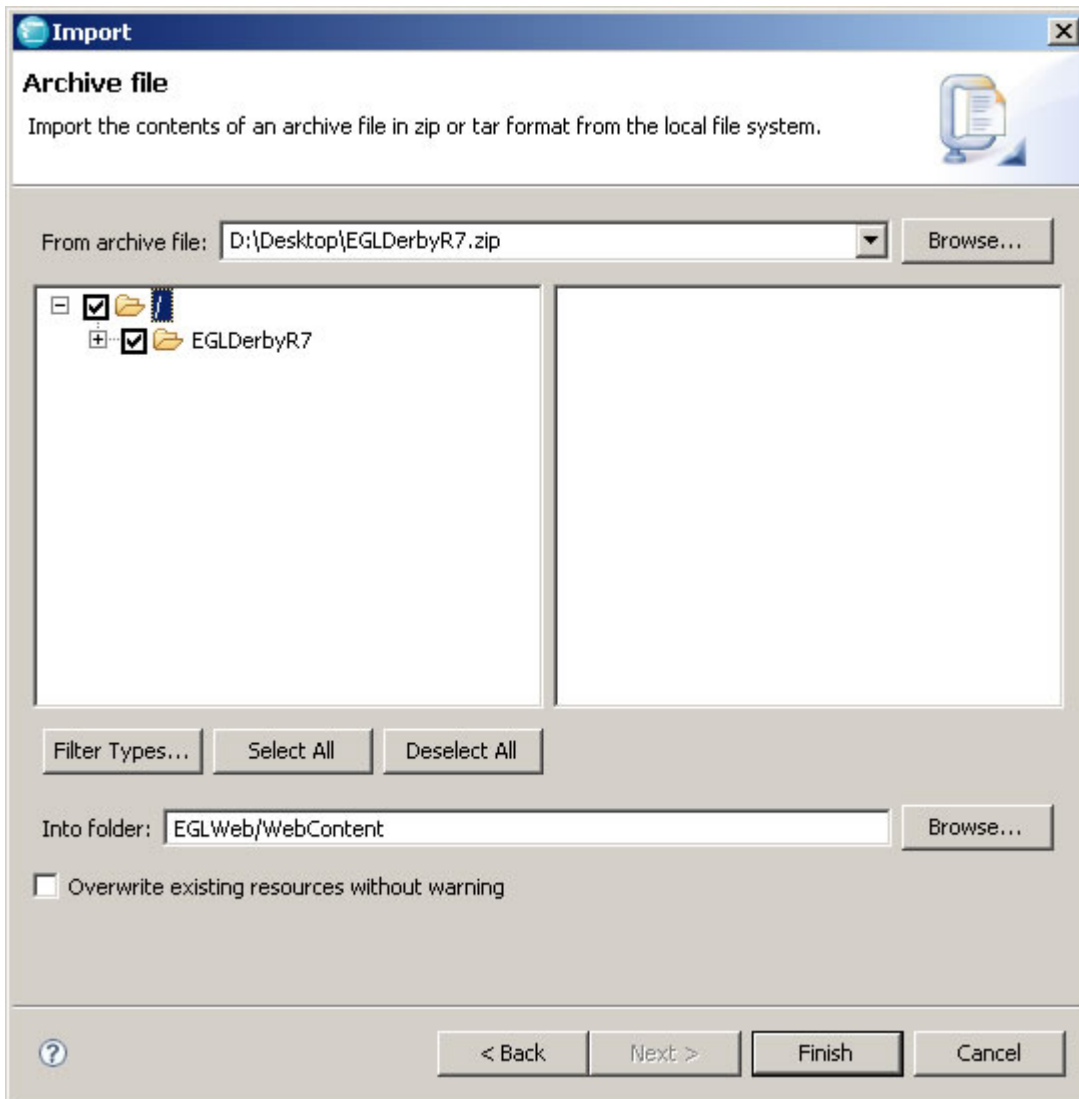
It doesn't matter where you save the database, as long as you can find it again later.

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_7.0.1/resources/EGLDerbyR7.zip`

Use your own Rational® Software Development Platform shared resources directory to replace `<shared_resources>`.

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



9. Click **Finish**.

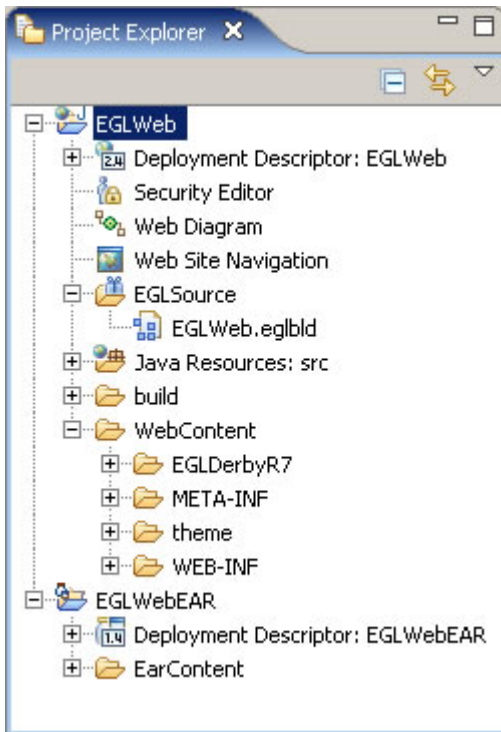
The database is added to your workspace in the WebContent folder of the EGLWeb project. Do not edit any of the files in the database directly. Later, you will create an EGL application to view and edit this database.

Lesson checkpoint

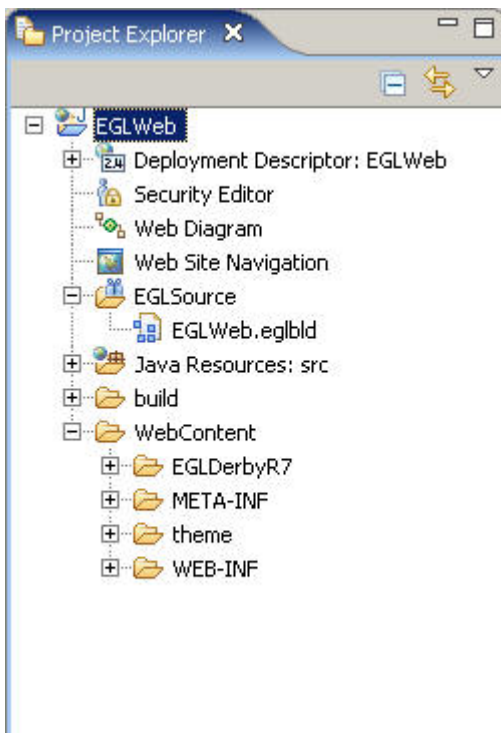
In this lesson, you created one or two projects, depending on your server. You can explore the project or projects in the Project Explorer view:

- 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 Project Explorer view looks like this if you are using WebSphere Application Server:



The Project 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 JNDI name that refers to your database and allows your EGL application to connect to it.

Recall that in the previous lesson, you created an EGL Web project and had it use the **JNDI name for SQL connection** jdbc/EGLDerbyR7. In this lesson, you configure the project so that this JNDI name refers to the sample database at run time.

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.

Setting up the database connection (WebSphere Application Server)

Show Me

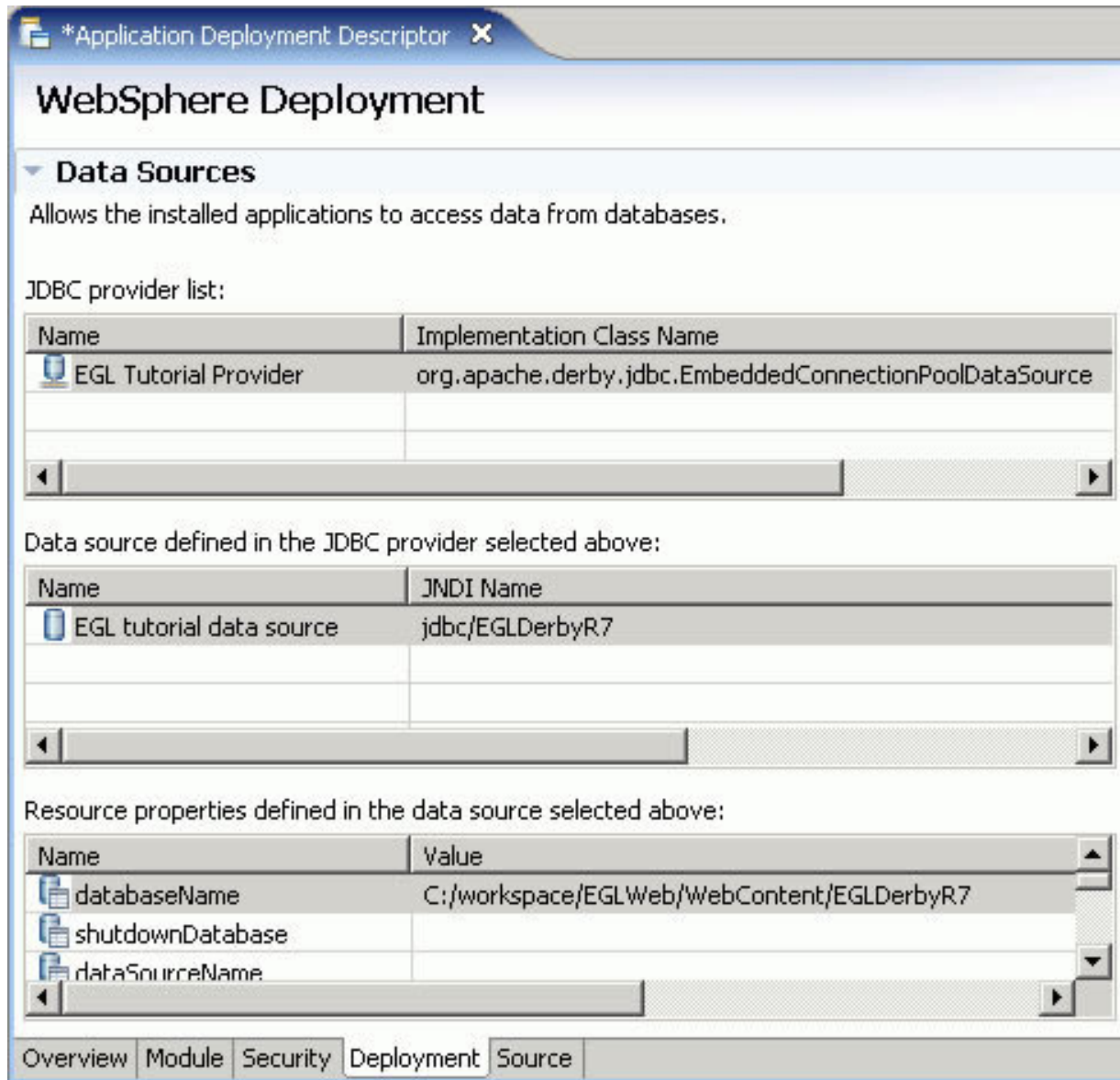
1. In the Project Explorer view, expand **EGLWebEAR** and double-click the **EGLWebEAR** deployment descriptor, which is directly underneath the project. The deployment descriptor for the EAR project opens in the editor. The deployment descriptor includes information on deploying the application, including information about how it will access data sources at run time.
2. At the bottom of the deployment descriptor editor, click the **Deployment** tab. The deployment descriptor editor shows the Deployment page.
3. Under **JDBC provider list**, click the **Add** button. This is the top **Add** button on the page. The Create JDBC Provider window opens.
4. In the **Database type** list, click **Derby**.
5. In the **JDBC provider type** list, select **Derby JDBC Provider**.
6. Click **Next**.
7. In the **Name** field, type the name EGL Tutorial Provider.
8. Click **Finish**. Now the new JDBC provider is listed under **JDBC provider list**.
9. Click the **EGL Tutorial Provider** to select it.
10. Under **Data source defined in the JDBC provider selected above**, click **Add**. This is the second **Add** button from the top of the page. The Create Data Source window opens.
11. In the **Select the type of JDBC provider** list, click **Derby JDBC Provider**.
12. Under **Select the data source type**, click **Version 5.0 data source**.
13. Click **Next**.
14. In the **Name** field, type the name EGL tutorial data source.
15. In the **JNDI name** field, enter the JNDI name for your project, exactly as shown:
jdbc/EGLDerbyR7

Note: The JNDI name is case-sensitive.

16. Click **Next**.
17. Under **Resource Properties**, click **databaseName**.
18. In the **Value** field, enter the location of the database folder on your computer, using forward slashes to separate the directories:
 - a. Open a folder window outside of the workbench.
 - b. Browse to the EGLDerbyR7 database folder in the EGLWeb project in your workspace.
 - c. Copy the location from the address bar of the folder window and paste it in the **Value** field in the Create a Data Source window.
 - d. Change every backslash (\) in the location into a forward slash (/).

For example, if your workspace is in the C:\workspace folder, you would copy and paste C:\workspace\EGLWeb\WebContent\EGLDerbyR7. When you change those backslashes to forward slashes, the string becomes C:/workspace/EGLWeb/WebContent/EGLDerbyR7.

19. Click **Finish**. The Deployment page of the deployment descriptor editor should look like this, with your own workspace information in the **databaseName** field:



20. Save and close the deployment descriptor.

Setting up the database connection (Tomcat)

To connect to a database in Tomcat, you must create a contexts file in the Web project that links the database on disk to the JNDI name. You must also add the Derby JAR file to Tomcat so it has the necessary tools to access the database.

1. In the Project Explorer view, expand EGLWeb/WebContent/META-INF.
2. Right-click the META-INF folder and then click **New** → **File**.
3. In the New File window, make sure that the EGLWeb/WebContent/META-INF folder is selected.
4. In the **File name** field, enter this file name, exactly as shown:
context.xml

5. Click **Finish**. The new file is created and opens in the editor.
6. In the editor for the context.xml file, click the **Source** tab at the bottom right of the editor.
7. Paste the following code into the empty file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
<Resource
  driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
  maxActive="4" maxIdle="2" maxWait="5000"
  auth="Container" name="jdbc/EGLDerbyR7"
  password="Administrator"
  type="javax.sql.DataSource"
  url="jdbc:derby:C:\workspace\EGLWeb\WebContent\EGLDerbyR7"
  username="Administrator"/>
</Context>
```

8. Update the url attribute of this code to specify the location of the database on your system. For example, if you have unzipped the database into D:\myworkspace\EGLWeb\WebContent\EGLDerbyR7 you would set the url line to


```
url="jdbc:derby:D:\myworkspace\EGLWeb\WebContent\EGLDerbyR7"
```
9. Save and close the file.
10. Add the file derby.jar to the common/lib folder in your Tomcat installation directory:
 - a. Locate the file derby.jar. There are several ways you may be able to find this file:
 - If you have installed IBM WebSphere Application Server, version 6.1, you can use the version of Derby that is included with the server. Look for derby.jar in the following folder:


```
<install_location>/runtimes/base_v61/derby/lib
```
 - If you have installed database tools along with your product, you may be able to find the file in the following location:


```
<shared_resources>/plugins/com.ibm.datatools.db2.cloudscape.driver_version/driver
```
 - If you can't find the file in the previous two locations, try searching for the derby.jar file in your product installation directory. Different installations may have the file in different locations.
 - If you can't find the file on your computer, you can download the file directly from the Derby Web site: <http://db.apache.org/derby/>. You will need to download the most recently released version of Derby and extract the derby.jar file to a place on your computer.
 - b. Put a copy of the file derby.jar into Tomcat's common/lib folder. For example, if you installed Tomcat to D:\Servers\tomcat5.5, you would put the derby.jar file into this location:


```
D:\Servers\tomcat5.5\common\lib
```

Remember where you put this file, because you will need it again later.

Now the JNDI name jdbc/EGLDerbyR7 is associated with the actual database and driver on your system so the server can access the database at run time.

Lesson checkpoint

In this lesson, you set up a database connection in one of two ways.

- If you are using WebSphere Application Server, you added the database connection information to the EAR project. Now, any of that project's modules can use the database connection.
- If you are using Tomcat, you added a contexts file to the EGL Web project that gives the information for the connection. The connection is valid only for this project, but you can copy the file to other projects as necessary.

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.

Connect to the database and create parts

Follow these steps to connect to the database and create the data parts to access that database:

Show Me

1. Click **File** → **New** → **Other**. The New window opens.
2. Expand **EGL** and click **EGL Data Access Application**.
3. Click **Next**.
4. On the Define project settings page, select **EGLWeb** in the **Project Name** list.
5. Next to **Database connection**, click **New**. The New Connection window opens.
6. On the Connection Parameters page, under **Select a database manager**, expand **Derby** and click **10.1**.
7. In the **Database location** field, select the following folder:
`<workspace-location>/EGLWeb/WebContent/EGLDerbyR7`, where `<workspace-location>` is the location of your workspace. You do not need to change the **User ID** or **Password** fields.
8. In the **Class location** field, enter the location of the file `derby.jar`. There are several ways you may be able to find this file:
 - If you have installed IBM WebSphere Application Server, version 6.1, you can use the version of Derby that is included with the server. Look for `derby.jar` in the following folder:
`<install_location>/runtimes/base_v61/derby/lib`
 - If you have installed database tools along with your product, you may be able to find the file in the following location:
`<shared_resources>/plugins/com.ibm.datatools.db2.cloudscape.driver_version/driver`
 - If you can't find the file in the previous two locations, try searching for the `derby.jar` file in your product installation directory. Different installations may have the file in different locations.
 - If you can't find the file on your computer, you can download the file directly from the Derby Web site: <http://db.apache.org/derby/>. You will need to download the most recently released version of Derby and extract the `derby.jar` file to a place on your computer.

The New Connection window looks like this, with your own workspace and location information in the **Database location** and **Class location** fields:

Connection identification

☒ Use default naming convention

Connection Name:

Select a database manager:

- + Cloudscape
- + DB2 UDB
- + DB2 UDB iSeries
- + DB2 UDB zSeries
- Derby
 - 10.0
 - 10.1
- + Informix
- + Oracle
- + SQL Server

JDBC driver:

Connection URL details

Database location:

☒ Create the database if required

☐ Upgrade the database to the current version

JDBC driver class:

Class location:

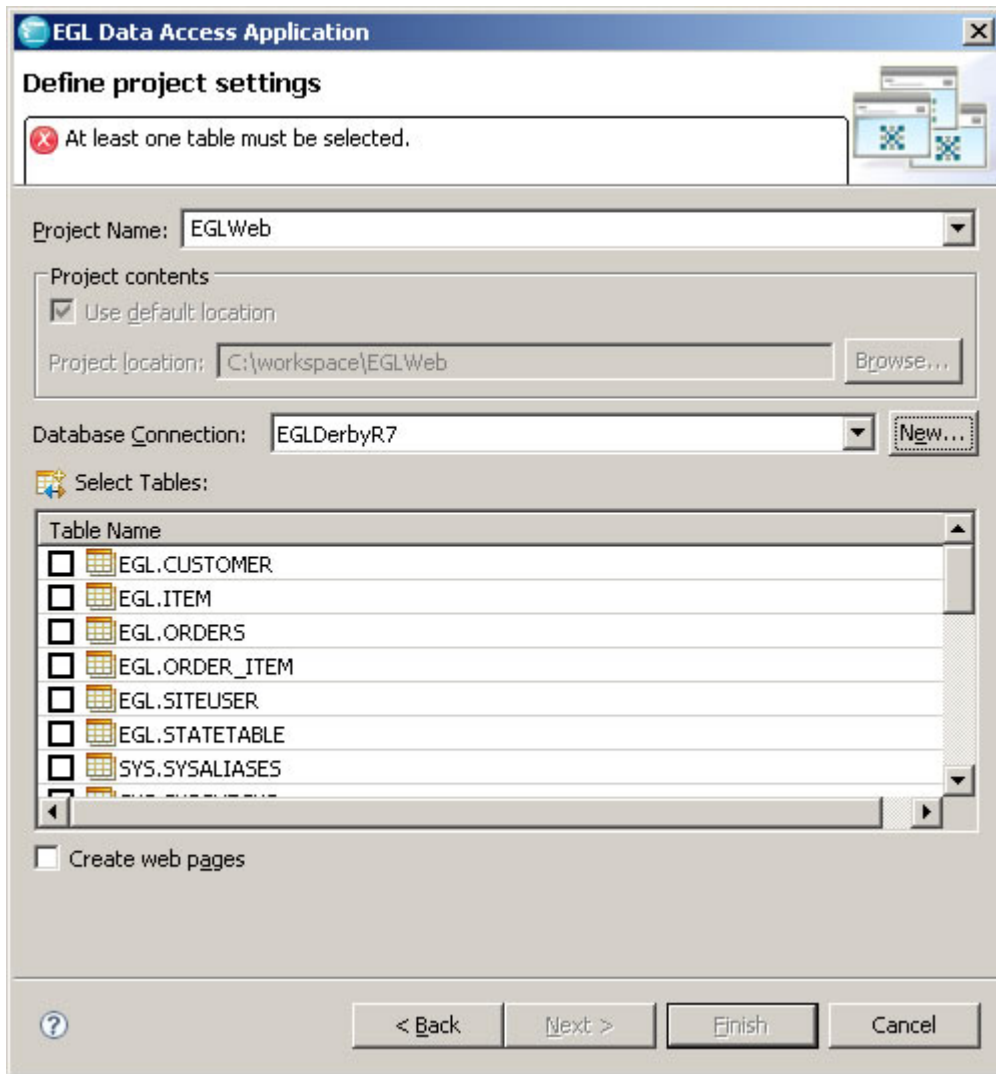
Connection URL:

User information

User ID:

Password:

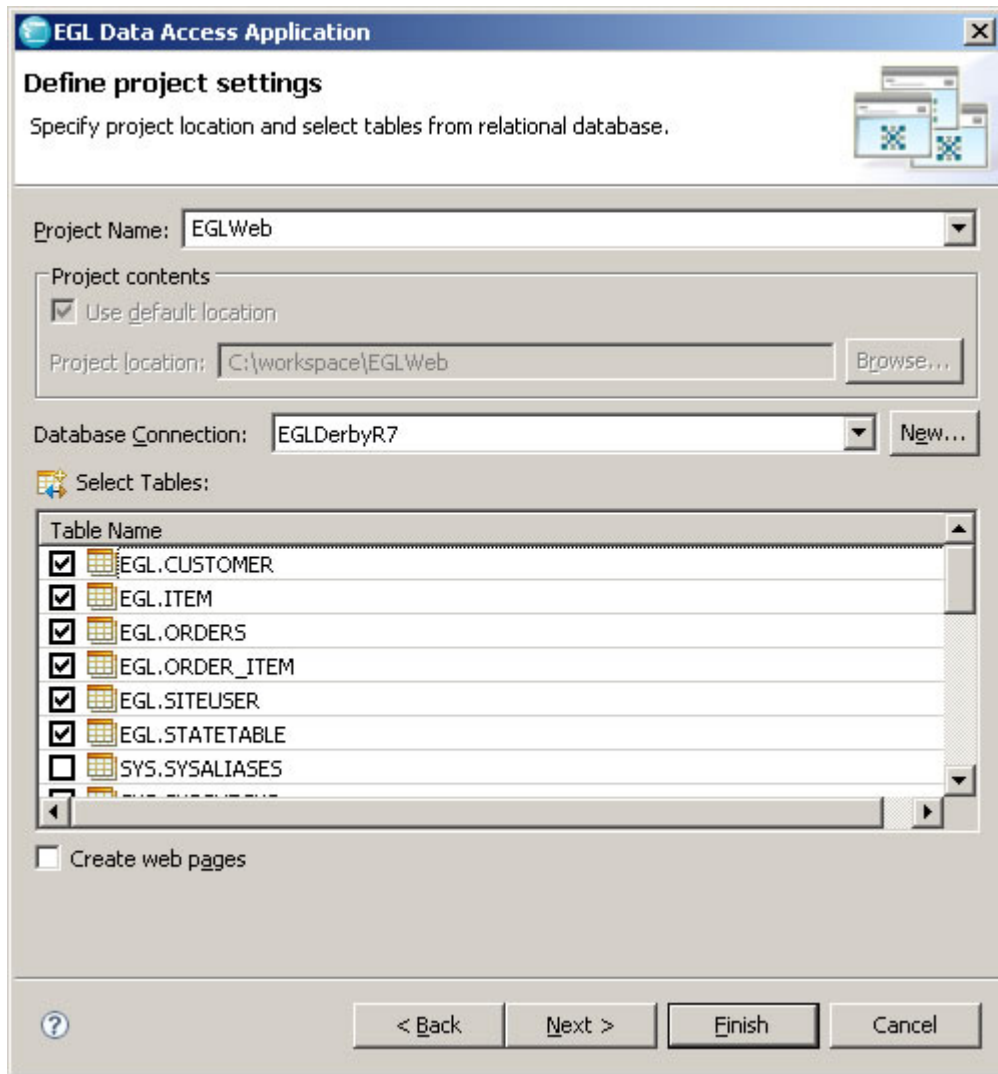
9. Click **Finish**. Now you have established a connection to the database. All of the tables in the database are listed under **Table Name** at the bottom of the wizard. You won't create data parts for all of these tables because some contain only metadata. The EGL Data Access Application window looks like this:



10. Under **Select Tables**, select the check boxes next to only the following tables:

- EGL.CUSTOMER
- EGL.ITEM
- EGL.ORDERS
- EGL.ORDER_ITEM
- EGL.SITEUSER
- EGL.STATETABLE

The EGL Data Access Application wizard looks like this:



11. Clear the **Create Web pages** check box. 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.
12. Make sure that your project **EGLWeb** is listed in the **Project Name** field.

Note: Don't click Finish yet! You need to change one more setting in this wizard.

13. 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.
14. Click **Next** again to move to the Define project creation options page.
15. On the Define project creation options page, select the **Qualify table names with schema** check box.
16. 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 `eglderbyr7.data`, `eglderbyr7.access`, and `eglderbyr7.primitivetypes.data`. Packages work just like folders: they contain your source code files and organize them into meaningful groups. In this case, the

eglderbyr7.data package holds the records, the eglderbyr7.access package holds the libraries, and the eglderbyr7.primitivetypes.data package holds the DataItems.

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

eglderbyr7.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.

eglderbyr7.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"],
                                   keyItems = [customerId]}

    customerId CustomerId {column = "EGL.CUSTOMER.CUSTOMER_ID";
    firstName  FirstName  {column = "EGL.CUSTOMER.FIRST_NAME";
    lastName   LastName   {column = "EGL.CUSTOMER.LAST_NAME";
    ...
end
```

eglderbyr7.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.

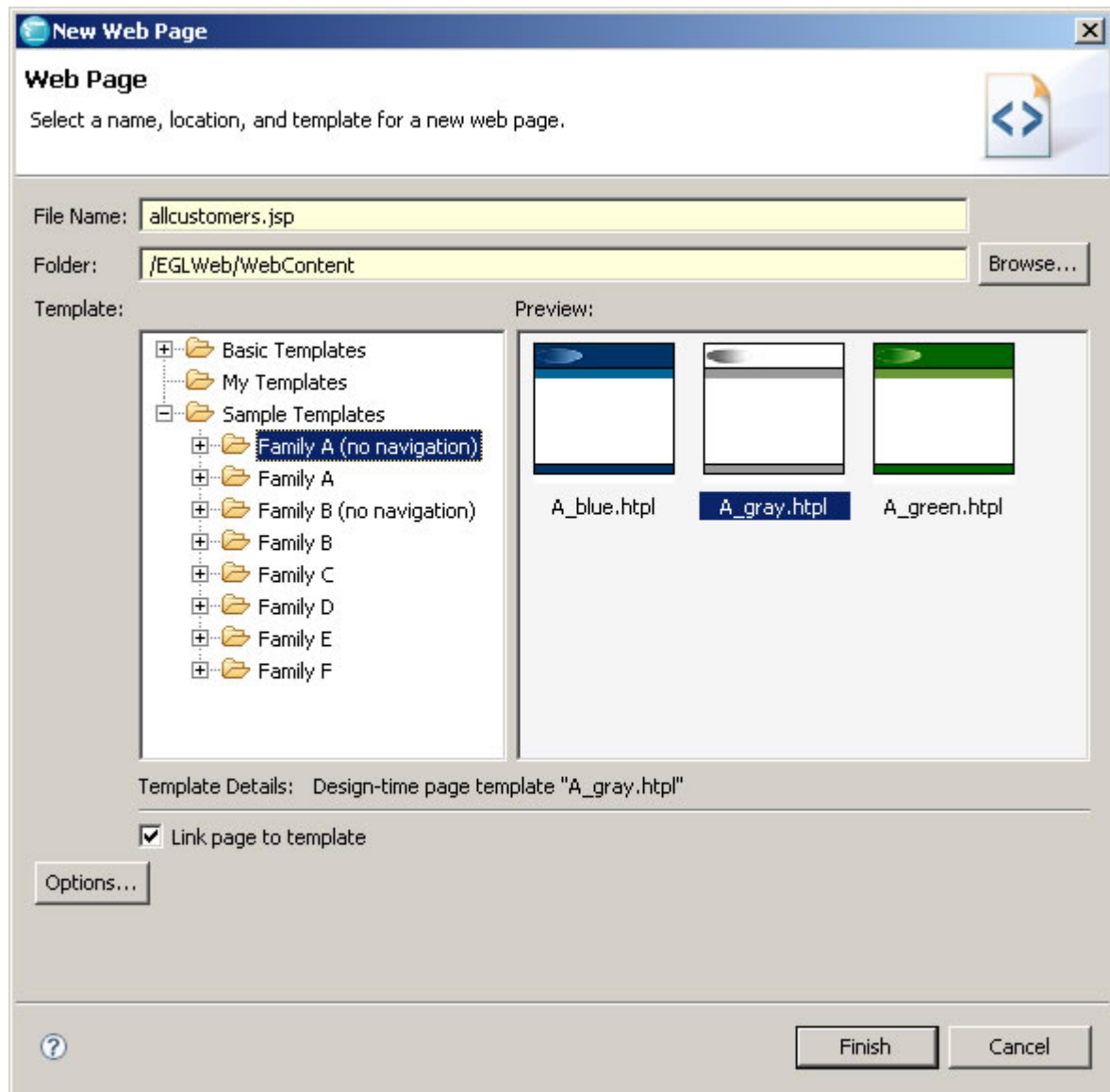
Create the JSP file from a template

Show Me

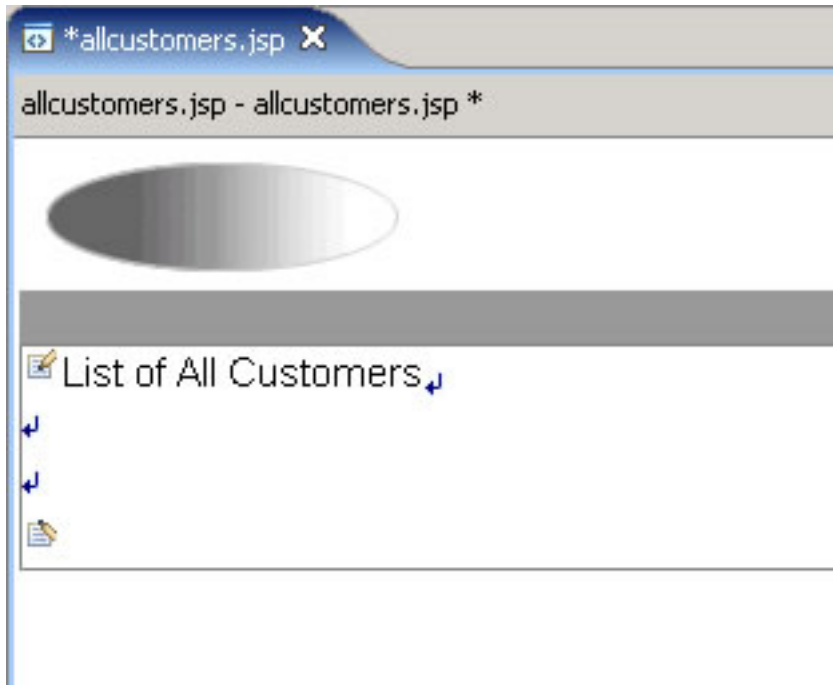
1. In the Project Explorer view, click the **WebContent** folder in the EGLWeb project to select it. It's important to select the place where you want to put new files, or else the new files might not appear in the place you expected.
2. Click **File** → **New** → **Other**.
3. In the New window, expand **Web** and click **Web page**.
4. Click **Next**. The New Web Page window opens.
5. In the **File Name** field, type this file name, including the extension:
allcustomers.jsp
6. Make sure that the **Folder** field lists the /EGLWeb/WebContent folder.
7. 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.

8. In the **Preview** box, click the **A_gray.html** template.

The New Web Page window looks like this:



9. Click **Finish**. The new Faces JSP file opens in the editor.
 10. In the new allcustomers.jsp file, remove the default text Place your page content here.
 11. In place of the default text, type this text:
List of all customers
 12. 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:



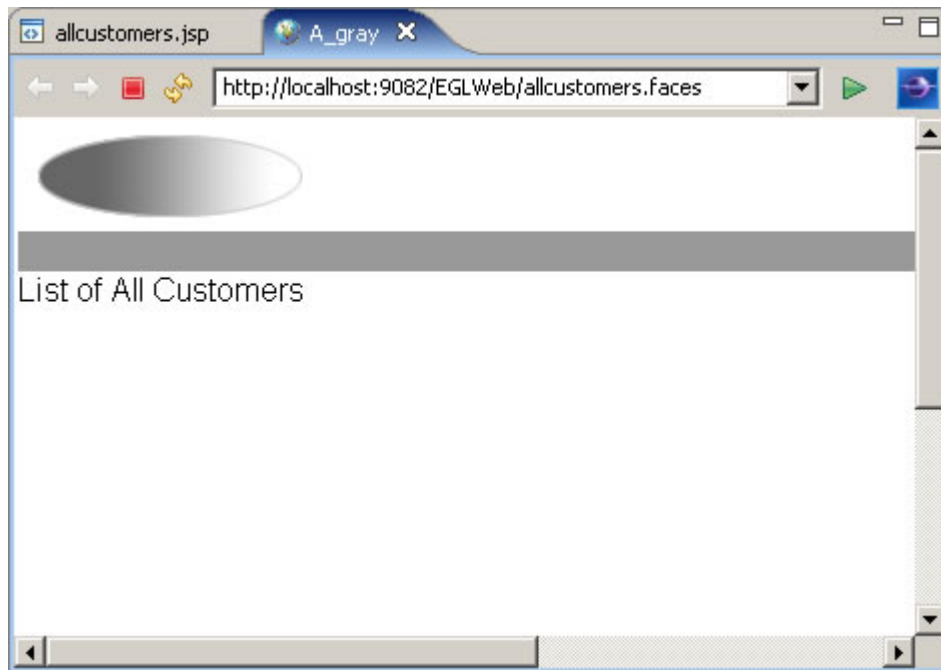
13. Save the file.

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.

1. In the Project Explorer view, right-click the **allcustomers.jsp** file. This file is in the WebContent folder of the EGLWeb project.
2. From the popup menu, click **Run As** → **Run on Server**. The Define a new server window opens.
3. In the Define a New Server window, select your server:
 - If you are using WebSphere Application Server, the server is shown in the list of servers. Click **Choose an existing server** and under **Select the server that you want to use**, click WebSphere Application Server.
 - If you are using Tomcat, you must add the server to this list:
 - a. Click **Manually define a new server**.
 - b. Expand **Apache**.
 - c. Click **Tomcat v5.5 Server**.
 - d. In the **Server Runtime** list, select **Apache Tomcat v5.5**.
4. Select the **Set server as project default (do not ask again)** check box.
5. Click **Finish**.

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



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.

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:

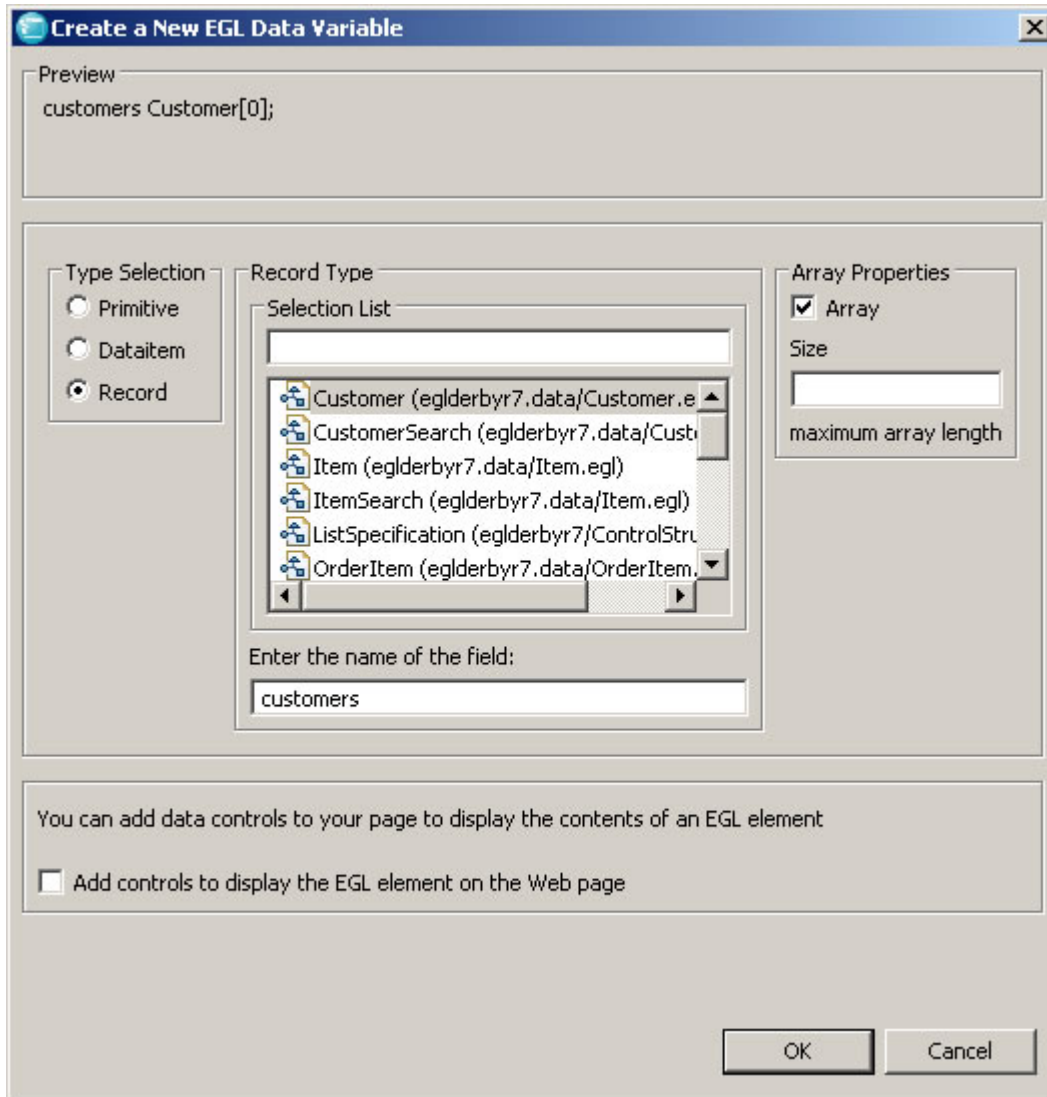
- You add fields to the Page Data view, associating those fields with the Web page.
- You place fields from the Page Data view onto the related Web page.
- You add a variable (in this case, an array of records) to an EGL JSF handler, which you can think of as the code that stands behind the run-time process. (In previous versions of EGL, a JSF handler was called a *page handler*.) 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.

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

Show Me

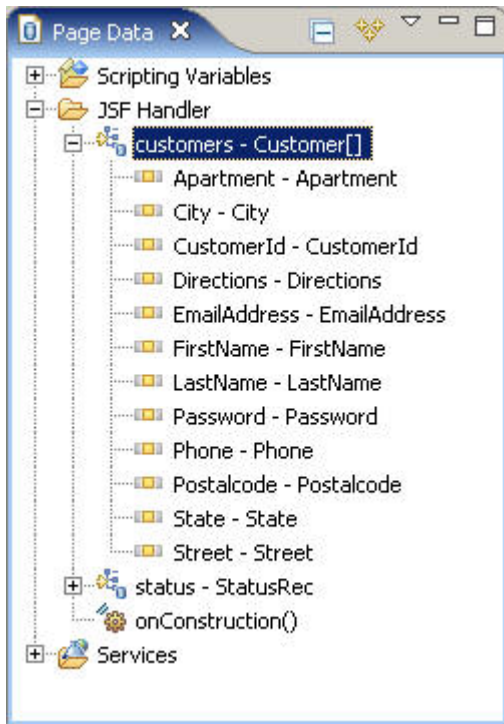
1. Open the **allcustomers.jsp** file by double-clicking it in the Project Explorer view.
2. Find the Page Data view, which is usually at the bottom left of the workbench. You can reveal the Page Data view by using the tabs, but if you can't find it, click **Window** → **Show View** → **Page Data**.
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 on 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**. An item representing the new variable appears in the Page Data view under **JSF Handler**.
12. Expand **JSF Handler** and then expand **customers - Customer[]**. There are 12 icons beneath this icon, 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. In the Page Data view, click the **customers - Customer[]** array variable to select it.
2. Click and 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 will be shown on the page.

3. Under **Create controls for**, click the radio button next to **Displaying an existing record (read-only)**.
With this option selected, the data on the page will be displayed in read-only output fields. If you choose **Updating an existing record**, the fields on the page will be input fields that you will be able to type into, and beneath the fields will be buttons for you to bind actions to. You'll create this type of fields 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
☒ CustomerId (int)	CustomerId	Output field
☒ FirstName (string)	FirstName	Output field
☒ LastName (string)	LastName	Output field
☐ Password (string)	Password	Output field
☐ Phone (string)	Phone	Output field
☐ EmailAddress (string)	EmailAddress	Output field

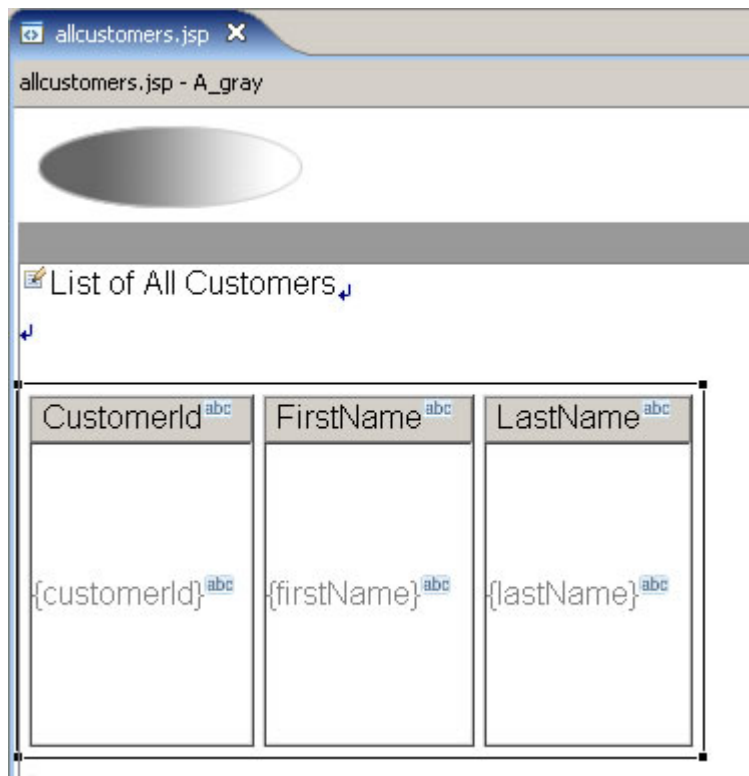
All None Options...

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:



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 on 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.

Add a function call to the EGL library

The next step is to add code to this page that invokes the 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 includes 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;`
5. On a blank line immediately after the first line of code, the line of code that begins package `jsfhandlers;`, add the following code, exactly as written:
`import eglderbyr7.StatusRec;`

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.

6. In the `allcustomers.egl` file, add a blank line between the lines `onConstructionFunction = onConstruction`, and `view = "allcustomers.jsp"`.
7. On this blank line, add the following code, exactly as written:

```
onPreRenderFunction = onPreRender,
```

The beginning of the JSF Handler part looks like this:

```
handler allcustomers type JSFHandler
{onConstructionFunction = onConstruction,
  onPreRenderFunction = onPreRender,
  view = "allcustomers.jsp"}
```

You've just added a *property* to the JSF Handler. In EGL, properties are name-value pairs that modify how a part behaves. Most types of EGL part can accept one or more properties, and each kind of part can accept different properties. In this case, the JSF Handler has three properties defined:

onConstructionFunction = onConstruction

The property **onConstructionFunction** specifies a function in the JSF Handler that runs the first time the JSF Handler's page is viewed in a browser. In this case, the property specifies a function named `onConstruction`, which is created by default in the JSF Handler. You won't be working with this function in this tutorial.

onPreRenderFunction = onPreRender

The property **onPreRenderFunction** specifies a function in the JSF Handler that runs each time the page is shown in the browser. Unlike the **onConstructionFunction**, the function defined in the **onPreRenderFunction** property runs each time the page is shown, including when the user refreshes the page or returns to the page after viewing another page. In the next few steps, you'll create this function and add code to retrieve current data from the database each time the page loads. The property specifies that the name of this function is `onPreRender`, and EGL will give an error if it can't find a function with this name in the part.

view = "allcustomers.jsp"

The property **view** 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.

8. Below the code `status StatusRec;`, add the following code:

```
function onPreRender()
end
```

9. Add a blank line between `function onPreRender()` and `end`.
10. On the new blank line, add the code to call the `GetCustomerListAll()` function to retrieve the data. In this case, try using the code completion ability of the EGL editor:
 - a. Place the cursor on the blank line between `status StatusRec;` and `end`.
 - b. Type the following code:

```
cust
```
 - c. Press **Ctrl+Spacebar**. The code completion window opens with all of the available EGL commands beginning with `cust`.
 - d. From the code completion 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 with the keyboard and pressing **Enter** or by

double-clicking it with the mouse. Be careful not to select the function `GetCustomerList(listSpecification, listOut Customer[], status StatusRec)`.

Now the new line of code reads `CustomerLib.GetCustomerListAll(customerArray, status);` and the `customerArray` parameter is highlighted.

- h. Change the default `customerArray` parameter 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:

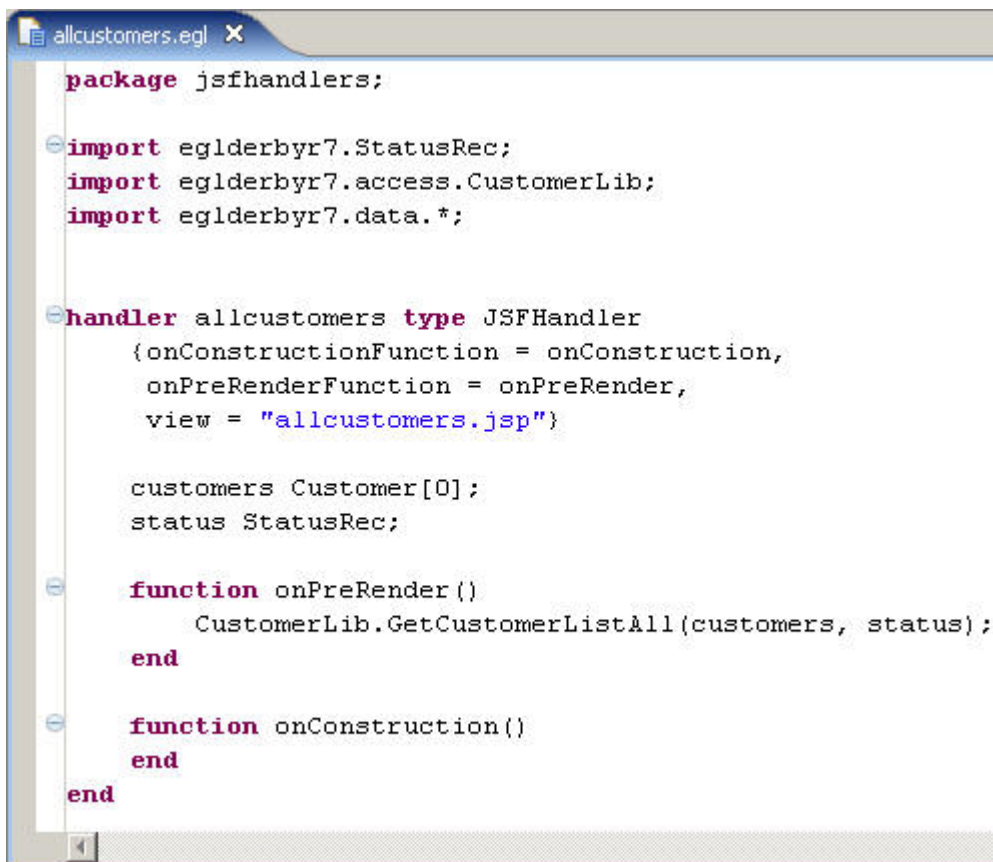
```
CustomerLib.GetCustomerListAll(customers, status);
```

Also note that there is a new `import` statement near the top of the file that reads `import eglderbyr7.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 refer to it directly.

The code completion method you used in the previous steps added this `import` statement automatically. If you had not used the code completion, you would have to either code this statement yourself or specify the explicit location of the library, making the line of code begin `eglderbyr7.access.CustomerLib.getCustomerListAll`.

- 11. Save the file.

Now, the `allcustomers.egl` file looks like this:



```
package jsfhandlers;

import eglderbyr7.StatusRec;
import eglderbyr7.access.CustomerLib;
import eglderbyr7.data.*;

handler allcustomers type JSFHandler
{onConstructionFunction = onConstruction,
 onPreRenderFunction = onPreRender,
 view = "allcustomers.jsp"}

customers Customer[0];
status StatusRec;

function onPreRender()
    CustomerLib.GetCustomerListAll(customers, status);
end

function onConstruction()
end

end
```

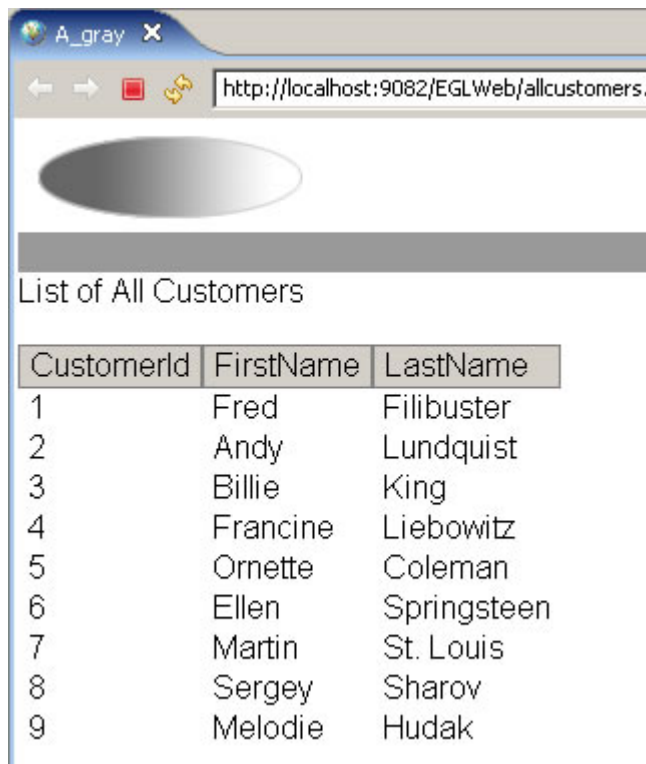
Here is the complete code of the `allcustomers.egl` file. If you see any errors marked by red X symbols in the file, make sure your code matches the code in this file: "Completed `allcustomers.egl` file after lesson 6" on page 41

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.

1. In the Project Explorer view, right-click the **EGLWeb** project and then click **Generate**.
2. In the Project 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. The page looks like this:



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 file **allcustomers.jsp**
2. In the Palette view, click the **Enhanced Faces Components** drawer to open it.
3. From the **Enhanced Faces Components** drawer, click the **Link** component to select it.
4. With the **Link** component selected in the Palette view, click directly on the `{lastName}` text field. 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.

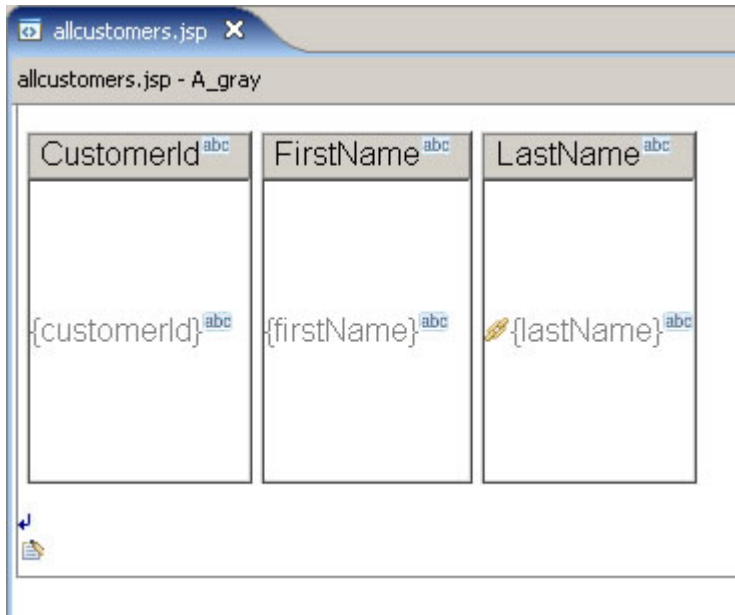
Leave the **Label** field blank. Without a label specified here, the link will use the text of the last name field itself as the text for the link.

6. Click **OK**.

If you see a link next to the {lastName} text field named **Link label**, you did not place the link directly onto the {lastName} text field. Click **Edit** → **Undo** and try again.

7. Save the page.

The page looks like this:



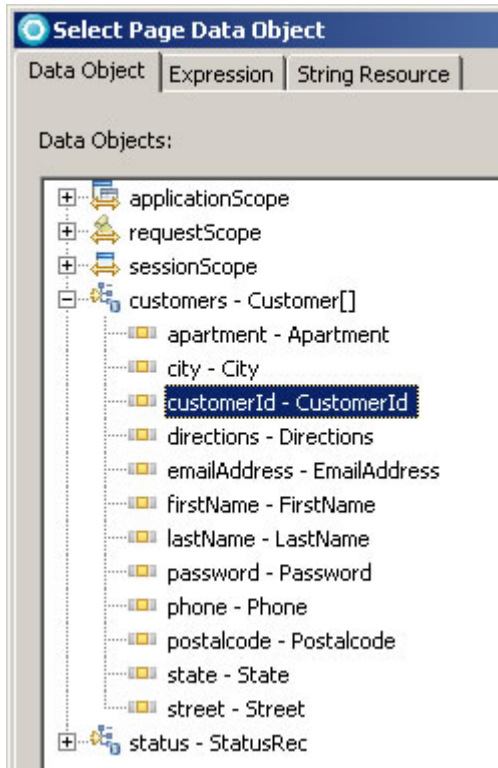
Add the parameter to the link

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 component you just added to the {lastName} text field.
The link icon itself , not the text field, 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 field. 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 on the **Name** column and type this text as the new name of the parameter:
CID
6. Click on the **Value** column.
7. Click the  **Select Page Data Object** button. The Select Page Data Object window opens.

8. Under **Data Objects**, click the + symbol to expand **customers - Customer[]**.
9. Click **customerId**.

The Select Page Data Object window looks like this:



10. Click **OK**.
11. Save the page.

Now, the value of the link's CID parameter is bound to the value of the customer_id field. When the user clicks on 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.

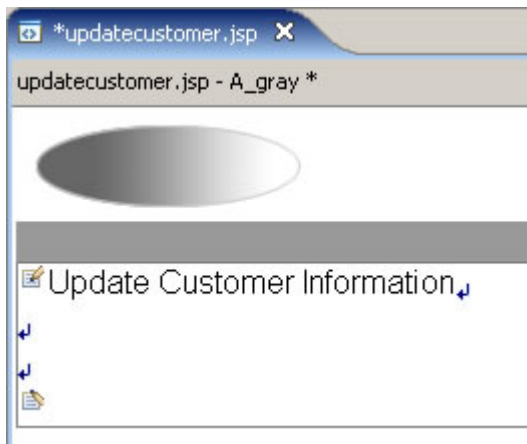
Create the file updatecustomer.jsp

Show Me

1. In the Project Explorer view, click the **WebContent** folder of the EGLWeb project.
2. Click **File** → **New** → **Other**.
3. In the New window, expand **Web** and click **Web page**.
4. Click **Next**. The New Web Page window opens.
5. In the **File Name** field, type this text as the name of the new file:
updatecustomer.jsp

6. Make sure that the **Folder** field lists the /EGLWeb/WebContent folder.
7. In the **Template** list, click **My Templates**.
8. In the **Preview** box, click the **A_gray.html** template.
9. Click **Finish**. The new page is created and opens in the editor.
10. Replace the default text with this text:
Update Customer Information
11. Press **Enter** three times to insert three blank lines.
12. 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} field on the page. This field does not mean that your page has errors; the {Error Messages} field marks the place where run time error messages will be displayed.

The page looks like this:

updatecustomer.jsp

updatecustomer.jsp - A_gray

Update Customer Information

CustomerId:	{customerId} abc
FirstName:	{firstName} abc
LastName:	{lastName} abc
Password:	{password} abc
Phone:	{phone} abc
EmailAddress:	{emailAddress} abc
Street:	{street} abc
Apartment:	{apartment} abc
City:	{city} abc
State:	{state} abc
Postalcode:	{postalcode} abc
Directions:	{directions} abc

Update this record {Error Messages}

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 new Web page's handler 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 file updatecustomer.jsp.
2. From the popup menu, click **Edit Page Code**. The updatecustomer.egl file opens in the editor.
3. In the updatecustomer.egl file, find the line `customer Customer;`
Like in the previous JSF handler you edited, you need to add a record to store the success or failure code of the SQL call.
4. Immediately after the line `customer Customer;`, add the following code, exactly as written:

```
status StatusRec;
```

5. On a blank line immediately after the first line of code, the line of code that begins package pagehandlers;, add the following code, exactly as written:

```
import eglderby7.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.

6. Just like on the allcustomers page, add the **onPreRender** attribute to the Handler, so the beginning of the Handler definition looks like this:

```
handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
  onPreRenderFunction = onPreRender,
  view = "updatecustomer.jsp"}
```

7. Just like you did in the allcustomers page, add the **onPreRender** function:

```
function onPreRender()
end
```

8. 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.

9. 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

10. 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 25.

```
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
```

11. If you did not use code completion to add the GetCustomer function, add the following import statement with the other import statement, immediately above the line that begins handler updatecustomer:

```
import eglderby7.access.CustomerLib;
```

12. Save the file.

The JSF handler looks like this:

```
updatecustomer.egl x
package jsfhandlers;

import eglderbyr7.StatusRec;
import eglderbyr7.access.CustomerLib;
import eglderbyr7.data.*;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
 onPreRenderFunction = onPreRender,
 view = "updatecustomer.jsp"}

customer Customer;
status StatusRec;

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

function onConstruction()
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 may want to use code completion or to copy the function from this page to make sure it is correct.

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

2. Save 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 can't find it there, double-click the updatecustomer.jsp file in the Project Explorer view, in the folder EGLWeb/WebContent/allcustomers.jsp.
4. In the Page Data view, expand **JSF Handler**. This folder lists all of the functions in the JSF handler except the onConstruction() function. In this case, this folder shows the updateRecord() function that you just created.

5. Drag and drop 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 41



```
package jsfhandlers;

import eglderbyr7.StatusRec;
import eglderbyr7.access.CustomerLib;
import eglderbyr7.data.*;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
 onPreRenderFunction = onPreRender,
 view = "updatecustomer.jsp"}

customer Customer;
status StatusRec;

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

function onConstruction()
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 Project Explorer view, right-click the `EGLWeb` project and then click **Generate**.
2. In the Project 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`.
3. 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.
4. Type a new `FIRST_NAME` for the record.

5. Type new information for a few of the other fields on this page. Do not change the CUSTOMER_ID field.
6. 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 that record's last name again to see the new information that was saved in the database.

Summary

This is the end of the tutorial *Introducing EGL (a quick-start guide)*.

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.

Other EGL tutorials:

Build a JSF search page with EGL

Create a hello world service with EGL

Create a hello world program with EGL

Resources

This tutorial used the following resources:

- A sample Derby database, found at the following location:
`<install_location>/eclipse/plugins/com.ibm.etools.egl.tutorial0001.doc_7.0.0/resources/EGLDerbyR7.zip`

Use your own Rational Software Development Platform installation directory for the `<install_location>`.

- "Completed `allcustomers.egl` file after lesson 6" on page 41
- "Completed `updatecustomer.egl` file after lesson 8" on page 41

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 eglderbyr7.data.*;
import eglderbyr7.StatusRec;
import eglderbyr7.access.CustomerLib;

handler allcustomers type JSFHandler
{onConstructionFunction = onConstruction,
  onPreRenderFunction = onPreRender,
  view = "allcustomers.jsp"}

customers Customer[0];
status StatusRec;

Function onPreRender()
  CustomerLib.GetCustomerListAll(customers, status);
End

Function onConstruction()
End

End
```

Return to “Lesson 6: Add data to the page” on page 25.

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 eglderbyr7.data.*;
import eglderbyr7.StatusRec;
import eglderbyr7.access.CustomerLib;

handler updatecustomer type JSFHandler
{onConstructionFunction = onConstruction,
  onPreRenderFunction = onPreRender,
  view = "updatecustomer.jsp"}

customer Customer;
status StatusRec;

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

function onConstruction()
end

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

End
```

Return to “Lesson 8: Create an update page” on page 34.