

Hello World: How to create a Java-based Web application without knowing Java technology

IBM Rational Business Developer and the business-oriented Enterprise Generation Language make it simple to create complex applications

Skill Level: Introductory

[Tim McMackin](#)
Software engineer
IBM

14 Sep 2007

Updated 22 Aug 2008

This tutorial demonstrates how to use IBM® Rational® Business Developer to create complex applications by using the Enterprise Generation Language (EGL), a simple business-oriented language, along with powerful graphical editing tools. The simple dynamic Web site that you will build for this example includes two pages: one to display a list of records in a database and another to allow users to change the data in one of those records. You can create this Java technology-based site without knowing any Java code nor the J2EE. You can download free trial versions of the software that you will need to work through this tutorial.

Section 1. Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this series

The [Hello World series](#) is for novice developers who want a high-level, hands-on overview of IBM software products. Each tutorial in the series provides simple exercises and step-by-step instructions to familiarize you with the components and use of a particular product. Upon completing a tutorial in the Hello World series, you will know enough about the product to begin exploring and using it on your own.

About this tutorial

In this tutorial, you learn how to use IBM Rational® Business Developer to build a simple dynamic Web site. This site that you build will have two pages: one to display a list of records in a database and another to allow users to change the data in one of those records.

Rational Business Developer is a key component of the IBM Rational Software Delivery Platform. It provides business-oriented developers accustomed to traditional procedural programming in languages, such as Report Program Generator (RPG) or COBOL, with the tools necessary to build complex and powerful modern applications. With Rational Business Developer, you can create applications with Web, Web service, character, or graphical interfaces; you can even create batch programs. You can then deploy these applications to Java™ application servers, to Java code, to COBOL on IBM® System i™, or to IBM® System z™ COBOL without knowing Java code, COBOL, or the intricacies of the server.

Enterprise Generation Language (EGL) is a programming language that you can use to write full-function applications quickly, thereby freeing you to focus on the business problem that your code is addressing rather than on software technologies. EGL is intended to be intuitive for people who are used to working with business-oriented languages, such as RPG, COBOL, and Visual Basic. It integrates with a wealth of Eclipse-based tools that are included with the Rational Business Developer to enable you to write code quickly, debug it in an agile and iterative fashion, and organize it clearly and effectively.

Objectives

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

System requirements

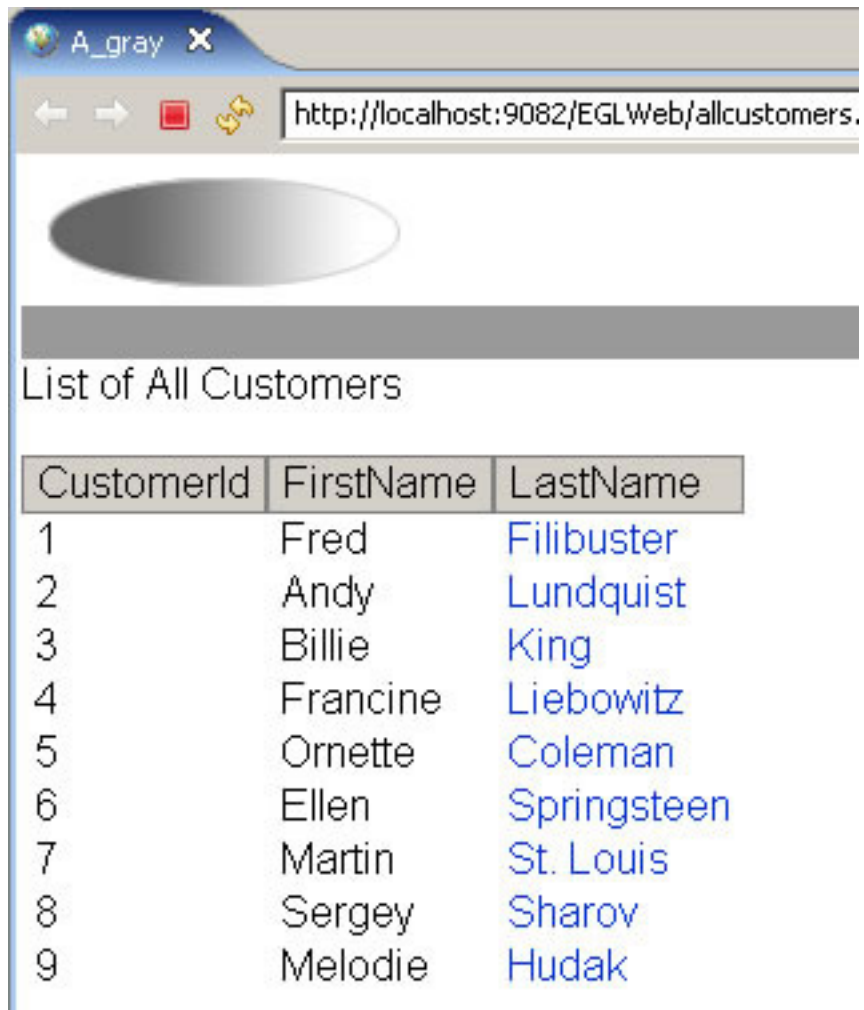
There are no prerequisites for this tutorial. However, to complete the steps, you need the following software installed on your computer before beginning:

- [IBM Rational Business Developer](#)
 - [IBM® WebSphere® Application Server](#) (included with the Rational Business Developer prerequisites)
-

Section 2. An introduction to the sample application for this tutorial

In this tutorial, you create an EGL Web project and import a sample database. You then create a simple EGL Web application that works with this database. The first of the two Web pages in the application, illustrated in Figure 1, shows a list of customers whose data is stored in the database.

Figure 1. Page that lists several rows from a database



The second Web page, illustrated in Figure 2, shows details about an individual customer and allows users to change those details.

Figure 2. Page that updates one row



A screenshot of a web browser window. The address bar shows the URL `http://localhost:9082/EGLWeb/updatecustomer`. The page content includes a large grey oval graphic, a horizontal grey bar, and the heading "Update Customer Information". Below the heading is a form with 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".

Rational Business Developer provides tools for creating this Web application, including hierarchical representations of the files in the application, graphical editing tools for Web pages, and editing tools for EGL.

In this tutorial, you will use EGL 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 selecting which page to present.
- On accepting the user's responses, those functions can apply additional business rules before storing the data.

Each of the two pages shown previously are controlled by EGL logic parts called *Handlers*, which control the runtime interaction with a user interface. In this case, the Handler parts are Java™ Server Faces (JSF) Handler parts, which are 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 can view and alter data stored in a database from a Web browser.

As you'll see in this tutorial, EGL promotes code reuse in several ways:

- First, EGL enables you to define *Dataltem parts*, which are a simple type of EGL data structure. A Dataltem 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 Dataltem to represent a telephone number. This Dataltem 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 Dataltem part.

Dataltem 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 Dataltem once and can use it as the basis for any number of variables or record fields.

- Second, EGL enables you to define *Record parts*, which are used as the basis for structured data. A Record part is a collection of other data parts (such as Dataltem 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 that represent 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 Dataltem parts, as you'll see in this tutorial. If you organize your data in this way, you can realize a more consistent definition of your data parts and increase efficiency, over time. Your changes to a single Dataltem part will cause a change in every variable that accesses the related, stored data.

- Third, EGL enables you to 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 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.
- Dataltem 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.

Section 3. Set up EGL


Important:

Before you can begin this tutorial, you must make sure that your system is configured to use EGL. Walk through the steps in this section 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 the capabilities that 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 "Enabling EGL capabilities" topic in the product Help system.

To enable EGL:

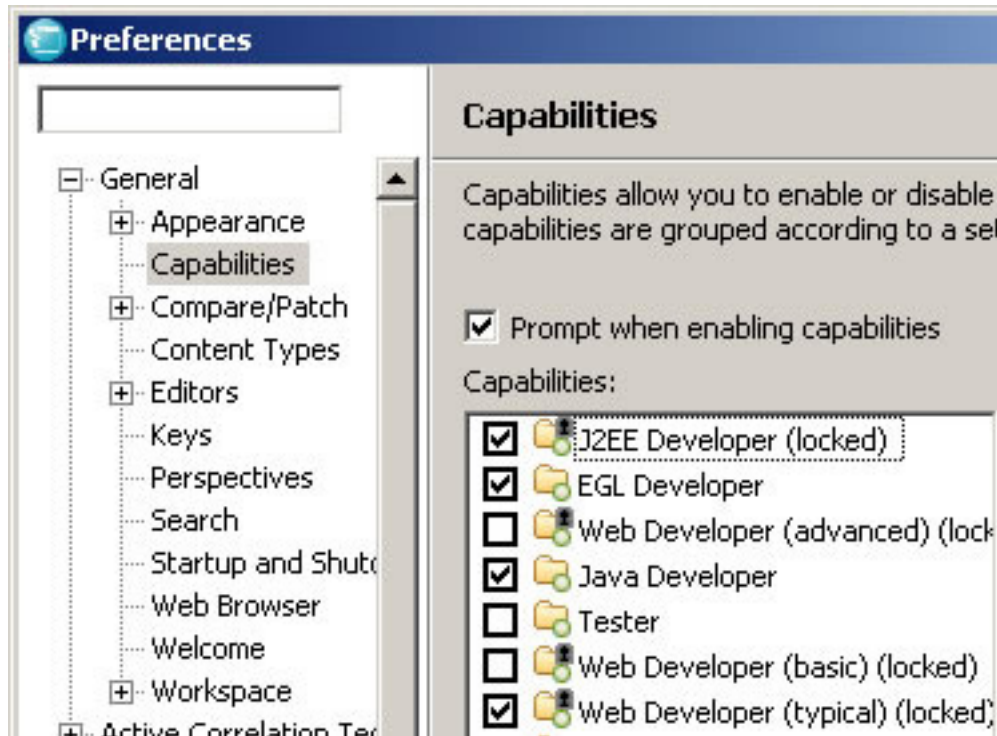
Would you like to see these steps demonstrated?

 [Show me](#)

1. You may want to use a separate workspace while working on the tutorial so you do not interfere with any of your other projects. To use a different workspace:
 - A. In the workbench, select **File > Switch Workspace** to open the Workspace Launcher window.
 - 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, choose **Window > Preferences** to open the Preferences window.
3. At the left side of the Preferences window, expand **General**.
4. Under General, select **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 should now look something like Figure 3. You may have capabilities available other than those illustrated in the figure, depending on the products and options you have installed.

Figure 3. Capabilities page of the Preferences window with the EGL Developer capability selected



6. Click **OK**.

You have now enabled the EGL capability necessary for creating EGL-related files and projects.


Tip:

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.

Prepare 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 page if it is open.
2. In the workbench, select **Window > Open Perspective > Other**.
3. When the Open Perspective window opens, select **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 selecting **Window > Open Perspective > Other** and then clicking a perspective. If you have closed or resized the views at any time, you can select **Window > Reset Perspective** to restore the perspective to its default settings.

Set generation options

To save time in large projects, EGL does not generate output from your code until you right-click the project and then select **Generate**. For this tutorial, you are working in two small logic parts called *Handlers*. You can set up the workbench to generate these Handler parts automatically each time that you save a change to a Handler:

1. In the workbench, select **Window > Preferences** to open the Preferences window.
2. In the Preferences window, expand **EGL** and select **Generation**.
3. On the Generate page, select (that is, add a check mark to) the check box labeled **Handler**.
4. Click **OK**.

Section 4. Create the projects and import the database

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

Your application needs two projects:

- The **EGL Web project** contains the EGL code, Web pages, and a sample database that make up the logic, data, and interface of the application. In this tutorial, you spend most of your time working with artifacts in an EGL Web project.
- The **Enterprise Archive project** (EAR project) contains information

about deploying an application in the Java® 2 Platform, Enterprise Edition (J2EE) framework, including information about 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.

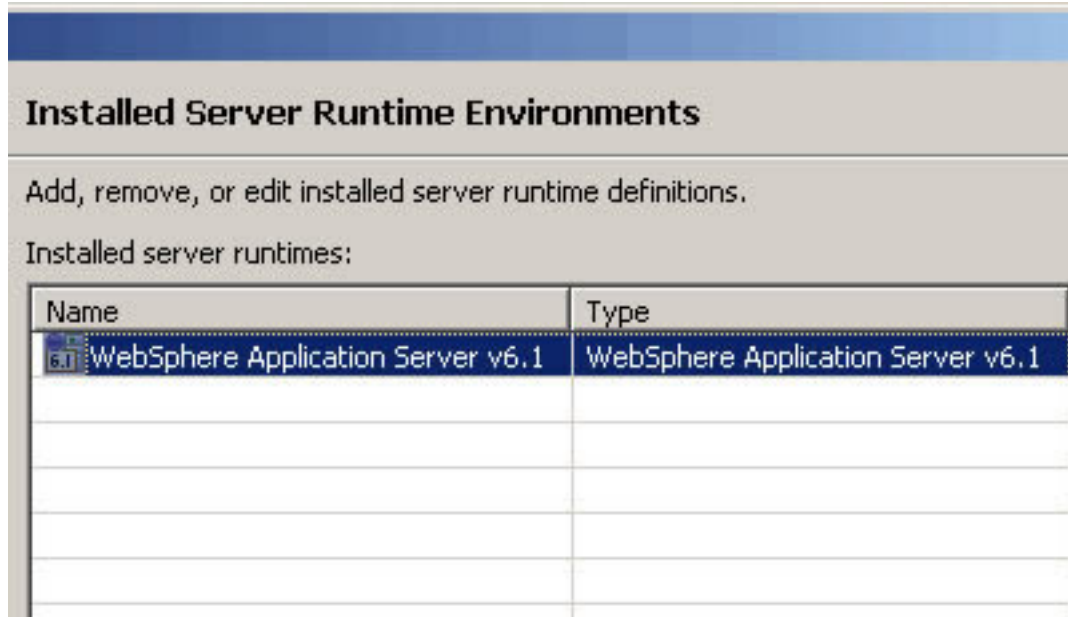
Sometimes the projects contained by an EAR project are called *modules*. In this case, your EGL Web project will be a module within the EAR project. In a large application, your enterprise application resource could have many modules doing different jobs.

Ensure that IBM WebSphere Application Server V6.1 is installed

To complete this tutorial, you need IBM® WebSphere® Application Server, Version 6.1, which you use to simulate an application server so that you can test your project.

1. From the menu, choose **Window > Preferences** to open the Preferences window.
2. At the left side of the Preferences window, expand **Server**.
3. Under Server, select **Installed Runtimes**. The right side of the Preferences window now shows a list of all of the server run times installed in the workbench.
4. Under Name, look for an entry named **IBM WebSphere Application Server v6.1**, as in Figure 4.

Figure 4. Installed run times in the Preferences window, showing the WebSphere Application Server runtime



5. If you see **IBM WebSphere Application Server v6.1** in the list, the test environment is installed. You may have other run times listed.
 - A. If you do not see an entry named "IBM WebSphere Application Server v6.1", you must run the product setup again.
 - B. When you run the product setup, select the check box to install IBM WebSphere Application Server, Version 6.1.
7. Click **OK**.

Create the EGL Web and EAR projects

Because your project will include 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. While creating the EGL Web project, EGL also creates an EAR file to go along with it.

Would you like to see these steps demonstrated?

 [Show me](#)

1. Select **File > New > Project** to open the New Project window.
2. Expand the **EGL** folder and select **EGL Project**.

3. Click **Next**.
4. In the Name field, enter `EGLWeb` as the name for your project.
5. Under EGL Project Types, select **Web Project**.
6. Click **Next**.
7. When the New EGL Web Project window opens, in the Target Runtime field, select **WebSphere Application Server v6.1**.
8. Under Build Descriptor Options, select **Create a new build descriptor**. You do not need to use the advanced settings unless you use WebSphere Application Server and you have previously changed the default setting that adds the project to an EAR. If you use WebSphere Application Server, your EGLService project must belong to an EAR project. The workbench retains this setting.
9. In the Java Naming and Directory Interface (JNDI) name for the SQL connection field, enter the following code, exactly as shown here:

```
jdbc/EGLDerbyR7
```

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

10. Select the **Add project to an EAR** check box.
11. In the Project Name field, enter `EGLWeb` as the name for your EAR project. The New EGL Web Project view should now look like Figure 5.

Figure 5. The New EGL Web Project window with the necessary fields filled in

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 Options...

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:

?

< Back
Next >

12. Click **Finish**.
13. If you see a message asking if you want to switch to the J2EE perspective, click **No**.
14. The Web diagram editor may open with a new Web diagram for your project. If this happens, close the Web diagram editor. The editor is used to determine the layout of a large site.

The new projects have now been created in your workspace.

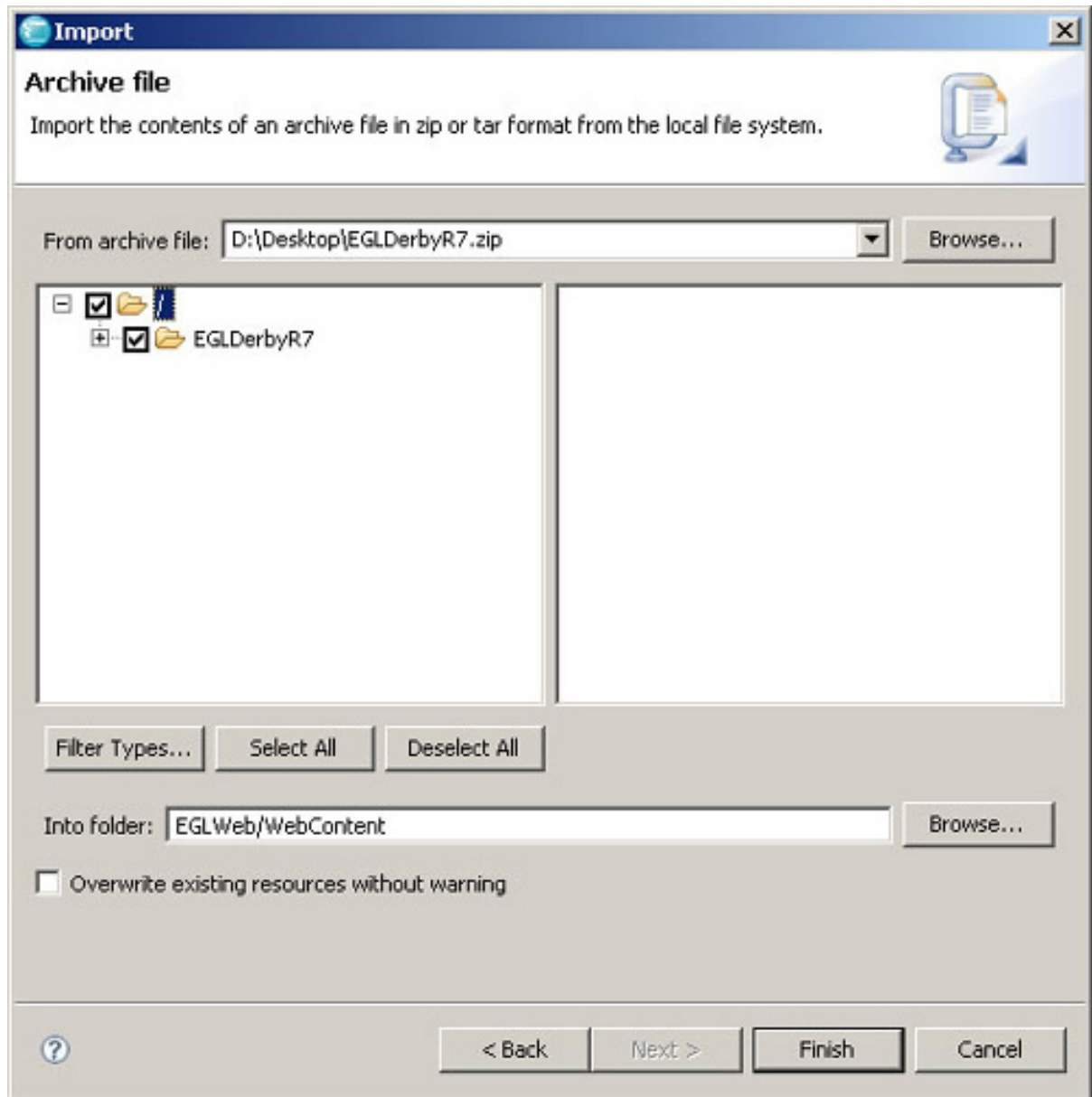
Import the database

This tutorial includes a sample Derby database to use in your application. The next steps add this database to your project. For more information on the Apache Derby open source relational database, see the link in [Resources](#).

1. Download the sample database (see [Downloads](#)) to a temporary folder on your computer, such as one on your desktop. It does not matter where you save the database, as long as you can find it again later. Alternatively, you can find the same database in the EGLDerbyR7.zip file in your product installation directory:
<shared_resources>/plugins/com.ibm.etools.egl.tutorial0001.doc_7.0.0/resources/directo
(Replace <shared_resources> with your own Rational Software Development Platform shared resources directory.)
2. In the workbench, select **File > Import**.
3. In the Import window, expand **General** and select **Archive File**.
4. Click **Next**.
5. In the "From archive file" field, enter the location of the file that you just downloaded. You can use the **Browse** button to find it.
6. At the bottom of the wizard view, next to the "Into folder" field, click the **Browse** button.
7. In the "Import into folder" view, 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 should look like Figure 6.

Figure 6. The Import window, importing the database into the project



9. Click **Finish**.

The database has now been added to your workspace in the WebContent folder of the EGLWeb project.

Note:

Do not edit any of the files in the database directly. Later, you will create an EGL application to view and edit this database.

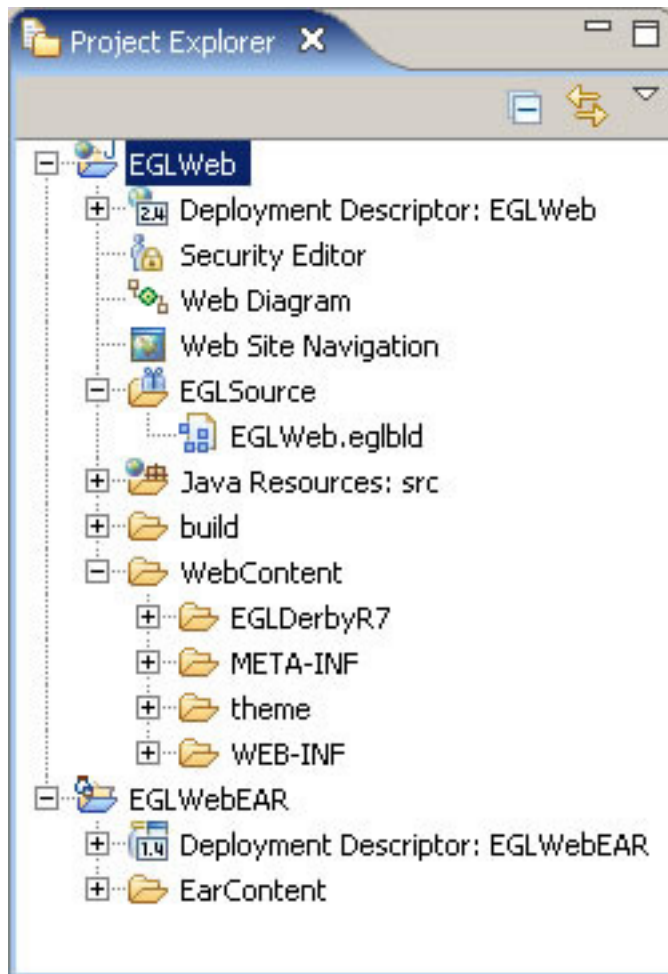
Checkpoint

In this section, you created two projects that you should be able to see 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 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 Archive file for the EGLWeb project. In the next section, you add the database connection information to this project so that your application can use the database.

Your projects should now look like Figure 7.

Figure 7. The Project Explorer view showing the EGL Web project and the database in the WebContent folder



Section 5. Set up the database connection

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


Because teaching you how to connect to a database is not the goal of this tutorial, this section does not explain the process in detail. In short, you set up a JNDI name that refers to your database and allows your EGL application to connect to it. WebSphere Application Server uses database connection information in the EAR project.

Recall that, in the previous section, you created an EGL Web project and used `jdbc/EGLDerbyR7` as the JNDI name for the SQL connection. In this section, you configure the EAR project so that this JNDI name refers to the sample database at

run time.

You must create a design-time connection to the database. When you use WebSphere Application Server, EGL automatically creates a matching runtime connection, as well. In the next lesson, you will also use this design-time connection to create starter EGL code.

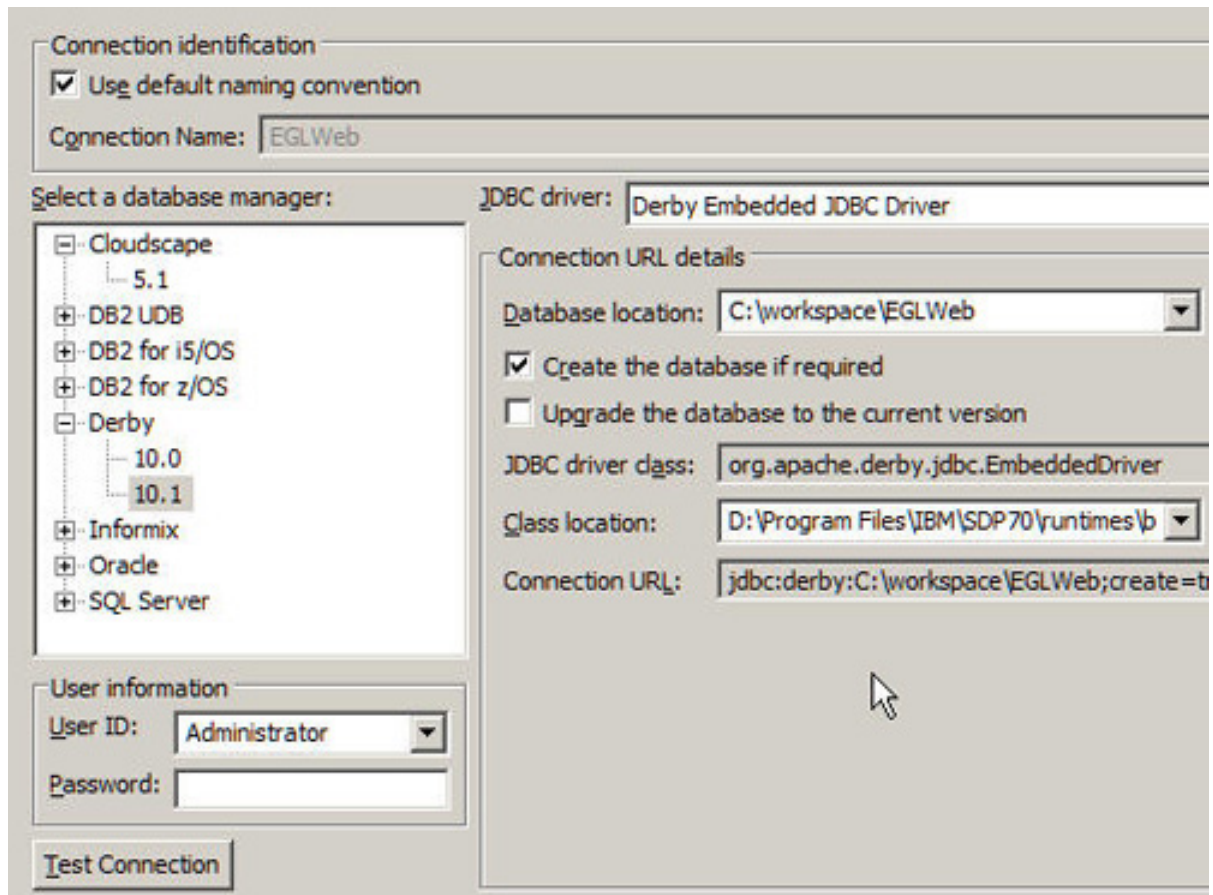
Would you like to see these steps demonstrated?

 [Show me](#)

1. In the Project Explorer view, right-click the EGLWeb project and then click **Properties**.
2. In the Properties window, click **EGL Runtime Data Source**.
3. In the EGL Runtime Data Source window, select **Load values from a data tools connection** and then click **New** to open the New Connection window.
4. In the New Connection window, under "Select a database manager", expand **Derby** and select **10.1**.
5. In the "Database location" field, click **Browse** and browse to this folder:
workspace-location/EGLWeb/EGLDerbyR7
where *workspace-location* is the full path to your workspace. You do not need to change the User ID or Password fields.
6. In the "Class location" field, enter the path to the derby.jar file:
install_location/runtimes/base_v61/derby/lib
where *install_location* is the path to the location of your installation.

The New Connection window should look like Figure 8, with your own workspace and location information in the **Database location** and **Class location** fields.

Figure 8. The completed New Connection fields



7. Click **Test Connection**. If the connection fails, verify that you have entered the correct information in the preceding steps and that your database is where you think it is.
8. Click **Finish**. The new connection is created and the necessary information for the connection is filled into the fields, as shown in Figure 9.

Figure 9. The completed EGL Runtime Data Source window

EGL Runtime Data Source

Enter the information that is used to connect to a database at runtime.

Load values from a data tools connection

Connection:

Input/modify values manually

Resource reference name:

JNDI name:

Data source requires a userid and password to obtain a connection

User ID:

Password:

Deploy database and connection properties when application is run on unit test server.

DB vendor name:

Data source classname:

Database name:

Class Location:

Database location (optional):

Server name (optional):

Port number (optional):

Most of this connection information comes from the data that you entered in the New Connection window. In addition, EGL has given this connection a JNDI name, which is an identifier for the connection. By default, the JNDI name is jdbc/EGLDerbyR7, based on the name of the database. The application will use this name to access the database connection at run time.

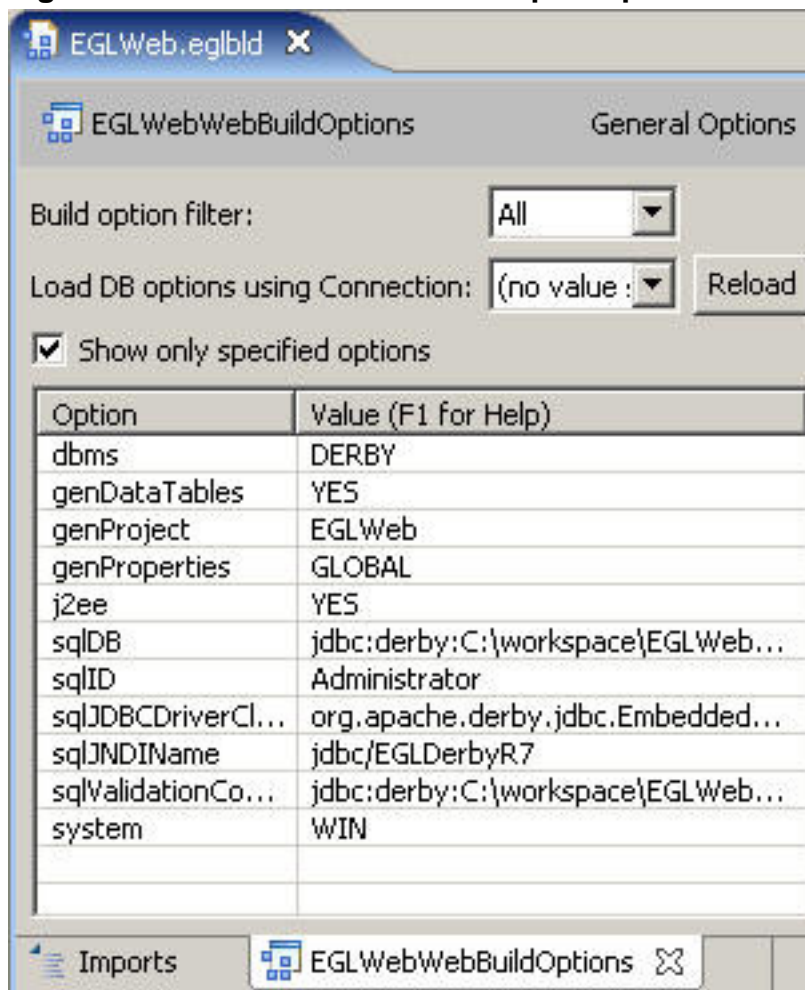
9. Click **OK**.
10. If you see a window asking if you want to update the information in the project's build descriptor options based on this connection, click **Yes**.
11. Open the build descriptor for the project by double-clicking the

EGLSource/EGLWeb.eglbid file in the Project Explorer view. The build descriptor opens in the build parts editor.

12. 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 those in Figure 10.

Figure 10. The correct build descriptor options



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

- **dbms**
Indicates the type of database, in this case DERBY.
- **sqlDB**
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, Java™ Database Connectivity, or JDBC)
- A colon separator
- The type of database (Derby)
- Another colon separator
- The path to the database on disk
- Any parameters for the connection

In this case, the connection URL is something like this example, with the path to your database in place of C:\workspace:

```
jdbc:derby:C:\workspace\EGLWeb\WebContent\EGLDerbyR7;create=true
```

- **sqlValidationConnectionURL**
Sets a connection URL to use to validate the connection to the database. In this case, as in most cases, this option has the same value as sqlDB.
- **sqlJDBCdriverClass**
Sets the name of the database driver, which is the program used to access the database. The New Connection window retrieved this name from the derby.jar file named org.apache.derby.jdbc.EmbeddedDriver.
- **sqlJNDIName**
The JNDI name that represents the connection at run time.

If the build descriptor options have been set based on the information that you added in the New Connection window, close the build descriptor without making any changes. If the build descriptor options have not been set, follow the steps in the sidebar to set them:

Set build descriptor options

1. In the **Load DB options using Connection** list, select your `EGLDerbyR7` connection. This action sets several of the options, except for the **sqlJNDIName** option.
2. Set the `sqlJNDIName` option to the following JNDI name, exactly as shown here:
`jdbc/EGLDerbyR7`

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.

3. Save your changes, and close the build descriptor.

13. *(Optional)* Set the EGL Runtime Database Connection to make these changes in the future by enabling the associated preference:
 - A. Click **Window > Preferences** and then click **EGL > Default Build Descriptor**.
 - B. 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.

14. In the Project Explorer view, expand **EGLWebEAR** and double-click the EGLWebEAR deployment descriptor, which is directly under 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.
15. At the bottom of the deployment descriptor editor, click the **Deployment** tab. The deployment descriptor editor shows the Deployment page.
16. Under JDBC provider list, click the **Add** button. This is the top Add button on the page.
17. When the Create JDBC Provider window opens, in the Database type list, select **Derby**.
18. In the JDBC provider type list, select **Derby JDBC Provider**.
19. Click **Next**.
20. In the Name field, type `EGL Tutorial Provider`.
21. Click **Finish**. Now the new JDBC provider is listed under the JDBC provider list.
22. Click the **EGL Tutorial Provider** to select it.
23. Under Data source defined in the JDBC provider selected above, click **Add**. This is the second **Add** button from the top of the page. .

24. When the Create Data Source window opens, in the "Select the type of JDBC provider" list, click **Derby JDBC Provider**.
25. Under "Select the data source type," click **Version 5.0 data source**.
26. Click **Next**.
27. In the Name field, type `EGL tutorial data source`.
28. In the JNDI name field, enter the JNDI name for your project, exactly as shown:

```
jdbc/EGLDerbyR7
```

Tip:

Remember that the JNDI name is case-sensitive.

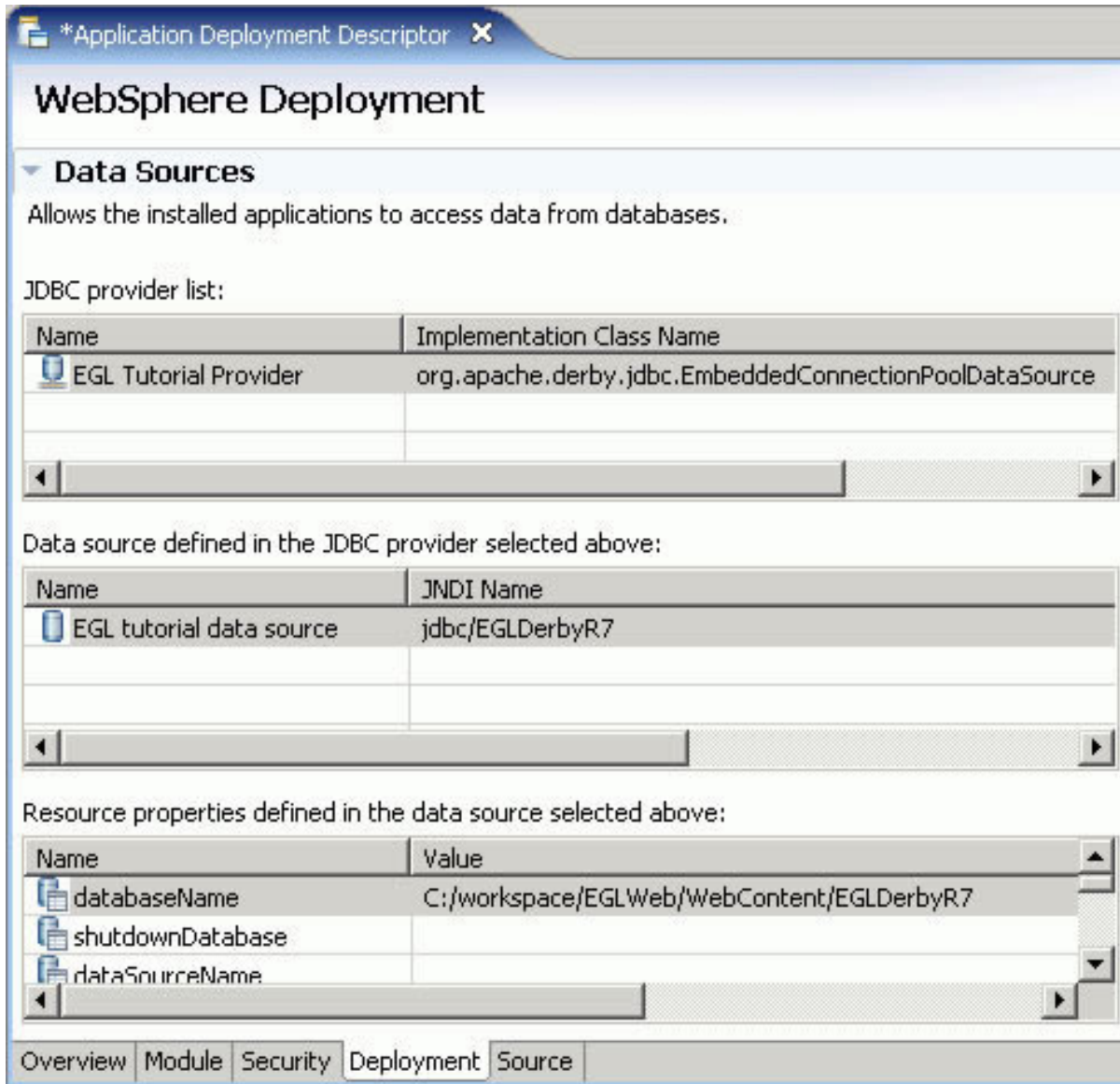
29. Click **Next**.
30. Under Resource Properties, select **databaseName**.
31. In the Value field, enter the location of the database folder on your computer, using forward slashes to separate the directories:
 - A. Open a folder window outside of the workbench.
 - B. Browse to the EGLDerbyR7 database folder in the EGLWeb project in your workspace.
 - C. Copy the location from the address bar of the folder window and paste it into the **Value** field in the Create a Data Source window.
 - D. Change every backslash (\) in the location to a forward slash (/). For example, if your workspace is in the `C:\workspace` folder, copy and paste:
`C:\workspace\EGLWeb\WebContent\EGLDerbyR7`. When you change those backslashes to forward slashes, the string becomes:

```
C:/workspace/EGLWeb/WebContent/EGLDerbyR7
```

33. Click **Finish**.

The Deployment page of the deployment descriptor editor should look like Figure 11, with your own workspace information in the `databaseName` field.

Figure 11. The Web deployment descriptor editor showing the JDBC provider created for the tutorial



33. Save your work, and close the deployment descriptor.

Checkpoint

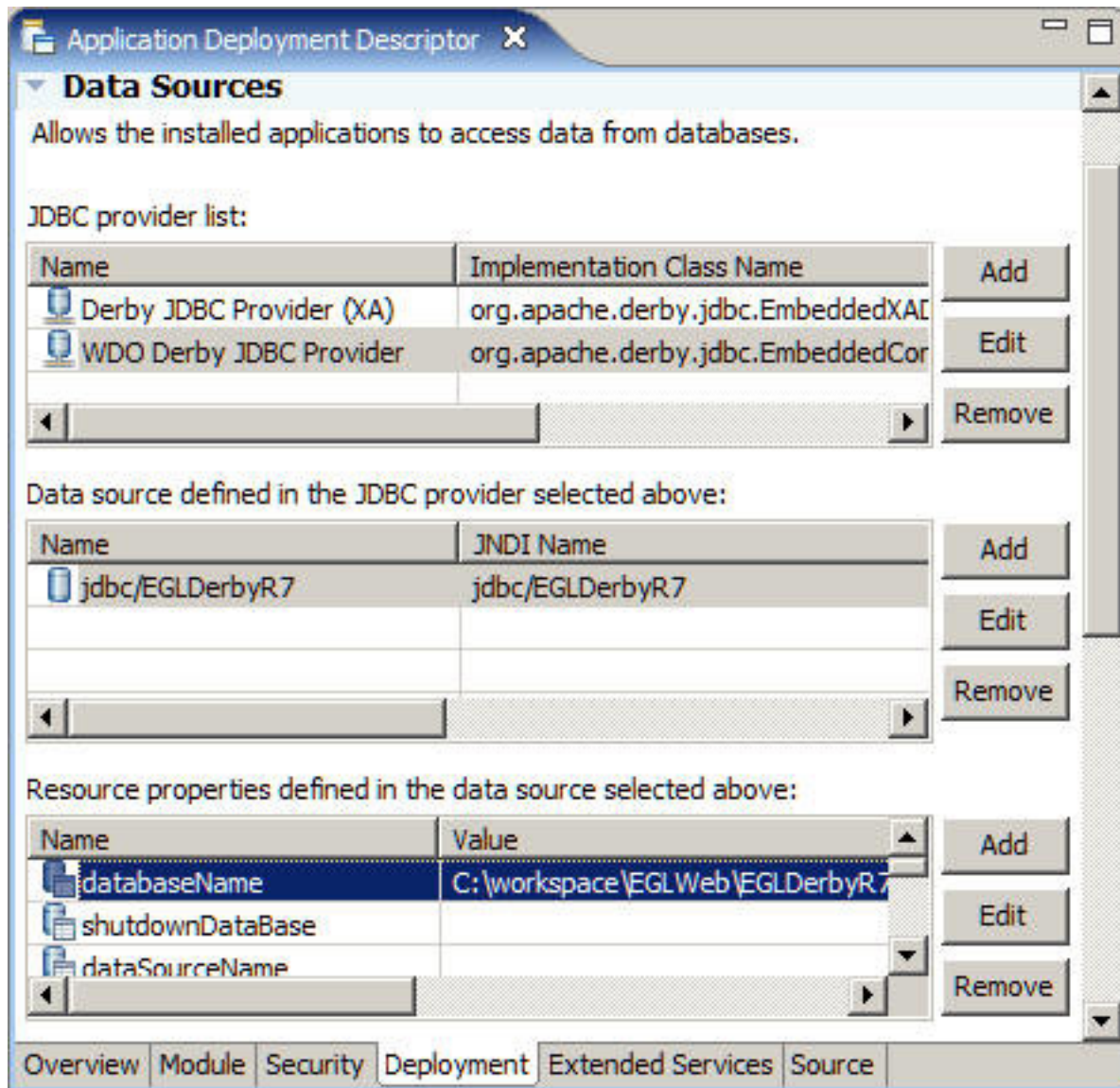
In this section, you set up a database connection for the project.

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 by using the workbench's data tools. Then EGL used the information in this design-time

connection to create a matching connection to be used at run time. The Enterprise Generation Language made the following changes to your projects:

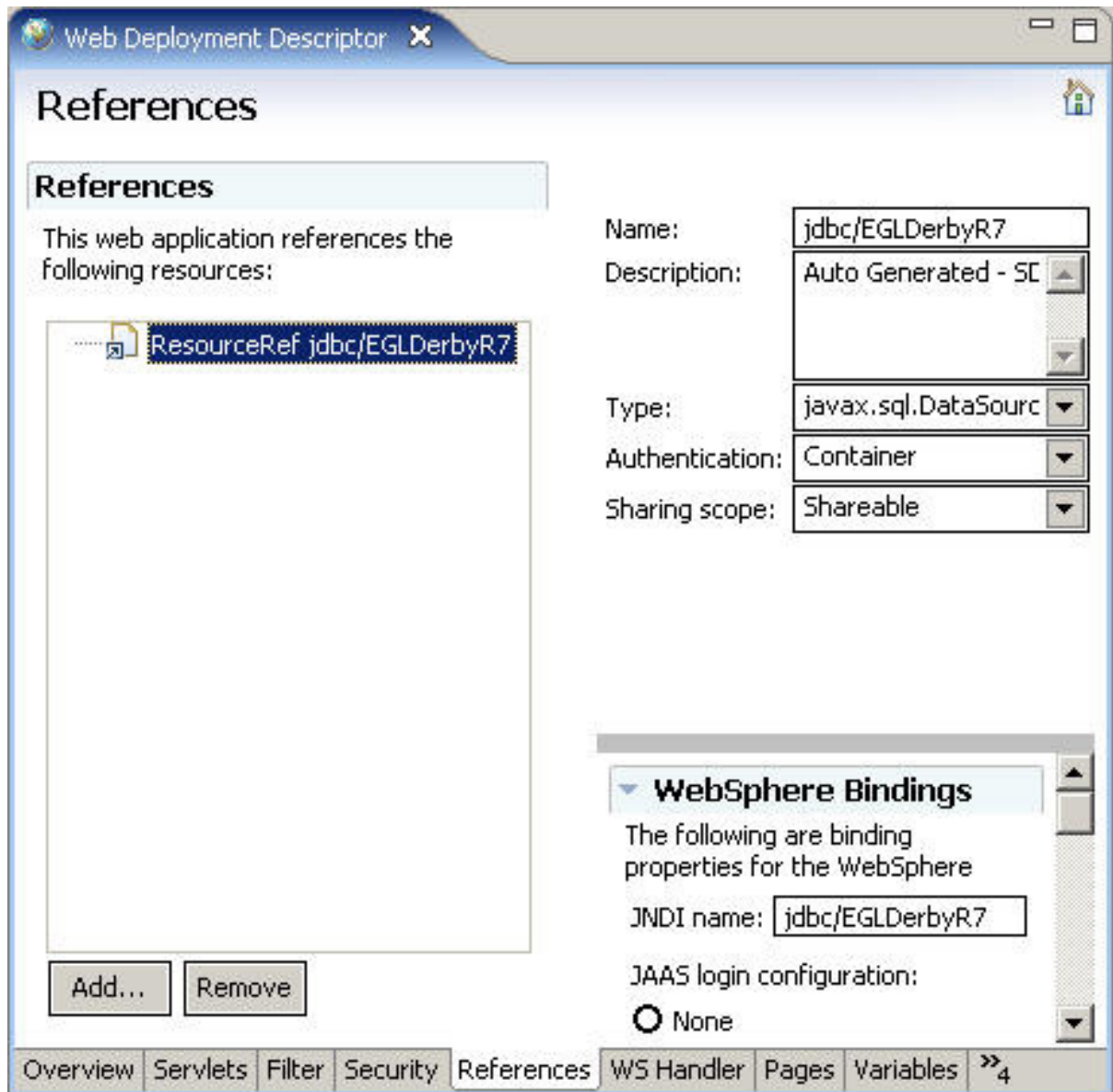
- EGL set the values of certain database-related build descriptor options, as explained previously.
- EGL created a JNDI name to use as a name for the connection. By default, the JNDI name created for your project is jdbc/EGLDerbyR7, based on the name of the database.
- EGL added a data source to the EAR project's deployment descriptor, as shown in Figure 12. This data source associates the JNDI name with the database itself. Now, other projects acting as modules within this EAR project can access the database through the JNDI name.

Figure 12. The EGL deployment descriptor for the EAR project



- EGL added a resource reference to that JNDI name in the EGLWeb project's J2EE Web deployment descriptor, shown in Figure 13. Now the EGLWeb project can use the data source defined in the EAR project by means of the JNDI name.

Figure 13. The EGL deployment descriptor for the EAR



Section 6. Create parts to access a database

Next, you 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, the wizard typically creates the following EGL parts:

- For each table that you select from the database, the wizard creates a *Record part* that represents that table. This Record part is a series of fields that represent each of the columns in the table.
- For each table that you select from the database, the wizard creates a *Library part*. This Library part 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 *Dataltem part*. These Dataltem parts represent the columns in the database. The Record parts created by the wizard consist of a sequence of these Dataltems.

Create parts from the database connection

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

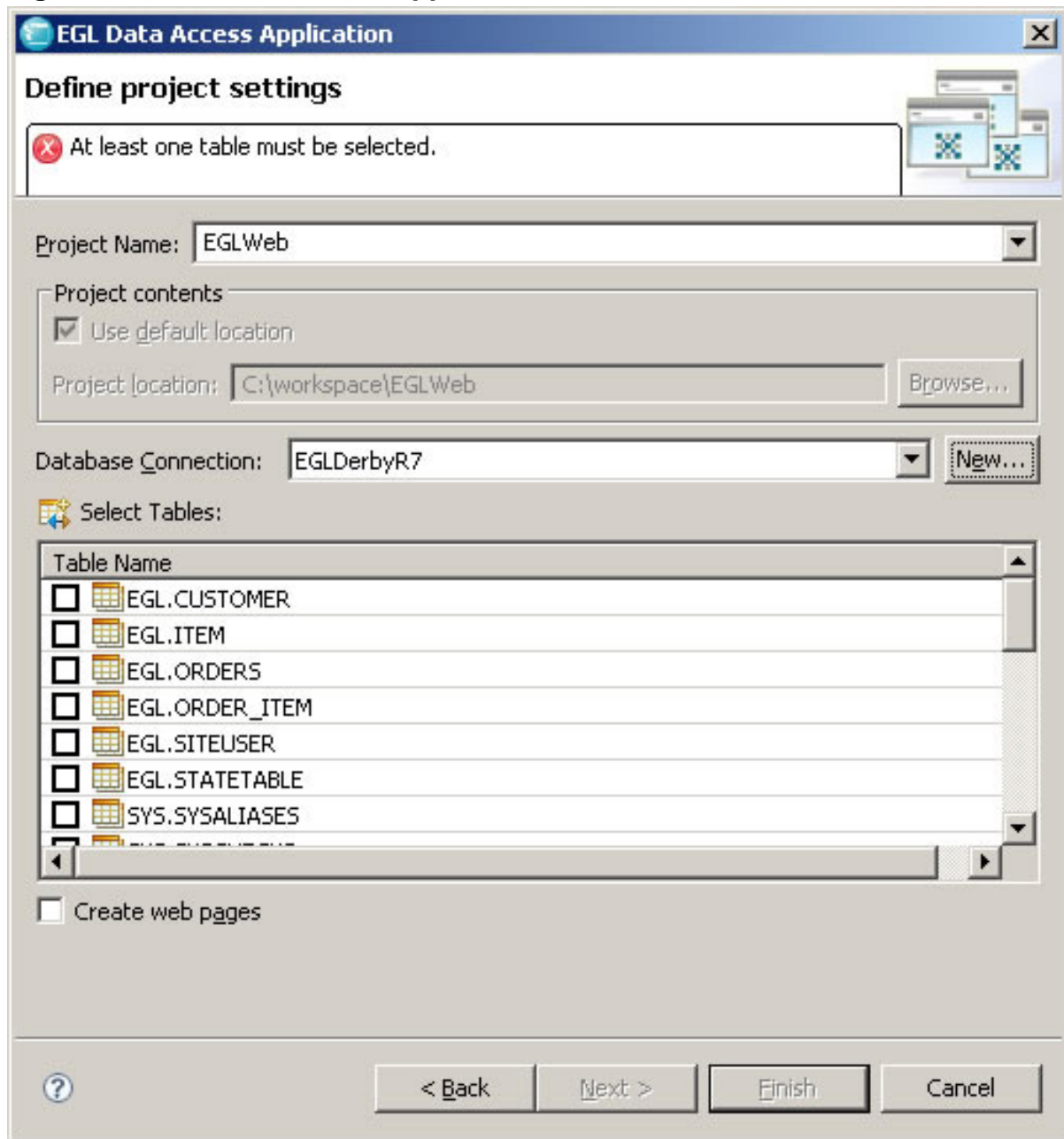
Would you like to see these steps demonstrated?



[Show me](#)

1. Click **File > New > Other** to open the New window.
2. Expand **EGL** and select **EGL Data Access Application**.
3. Click **Next**.
4. On the **Define project settings** page, select **EGLWeb** in the **Project Name** list.
5. From the **Database Connection** list, select the **EGLDerbyR7** connection that you created previously. You may be prompted for a User ID and password for the database. If so, enter whatever you want in those fields; the database does not require a password, but the workbench data tools expect a password.
6. 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 should look like Figure 14.

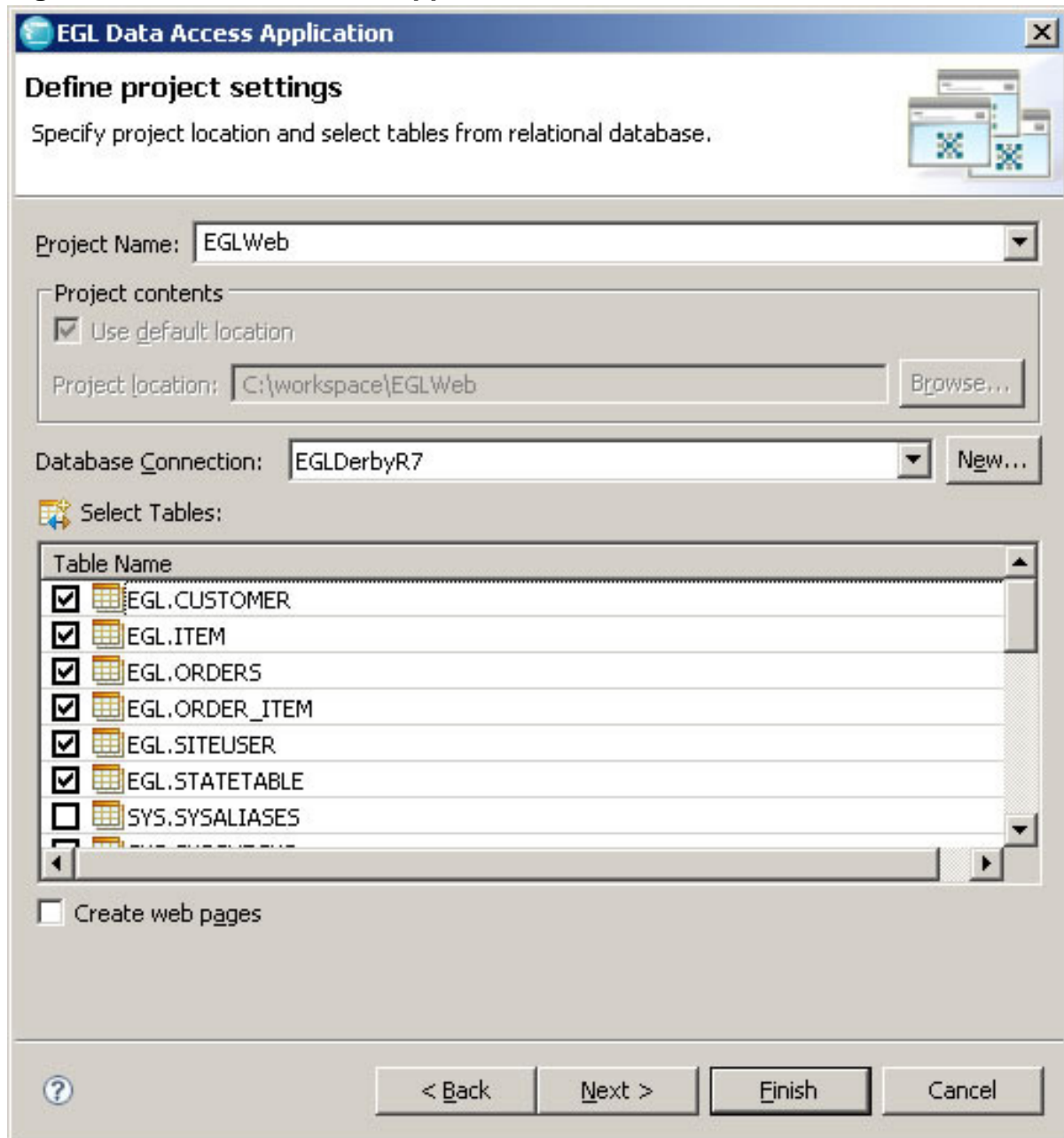
Figure 14. EGL Data Access Application window with no tables selected

7. Under Select Tables, select the check boxes next to the following tables:
 - EGL.CUSTOMER
 - EGL.ITEM
 - EGL.ORDERS
 - EGL.ORDER_ITEM

- EGL.SITEUSER
- EGL.STATETABLE

The EGL Data Access Application wizard should now look like Figure 15.

Figure 15. EGL Data Access Application window with the six tables selected



8. 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 having it do so now would defeat the purpose of this tutorial.

9. Make sure that EGLWeb (your project) is listed in the Project Name field.
10. **Do not** click **Finish** yet! You need to change one more setting in this wizard. Click **Next** to move to the "Define the fields" page, where you can 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.
11. Click **Next** again to move to the "Define project creation options" page.
12. On the "Define project creation options" page, select the **Qualify table names with schema** check box.
13. Click **Finish**.

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:

- `eglderbyr7.data` package holds the records
- `eglderbyr7.access` package holds the libraries
- `eglderbyr7.primitivetypes.data` package holds the Dataltems

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

- **`eglderbyr7.primitivetypes.data.DataDefinitions.egl`**: This file lists all of the Dataltems 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 Dataltem named `CUSTOMER_ID`:

```
dataitem CustomerId INT end
```

In this case, the ID number field is based on the integer primitive type. A Dataltem can have other properties to specify details, such as 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 -- the Customer table, in this case. This record includes fields for 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.

Section 7. Create a Web page

Now, create a Web page in the form of a Faces Java™ Server Pages (JSP) file. In the next section, you add data to this page by using the data parts and functions that you created previously. When the page is finished, it shows a list of every record in the database.

Create the JSP file from a template

Would you like to see these steps demonstrated?

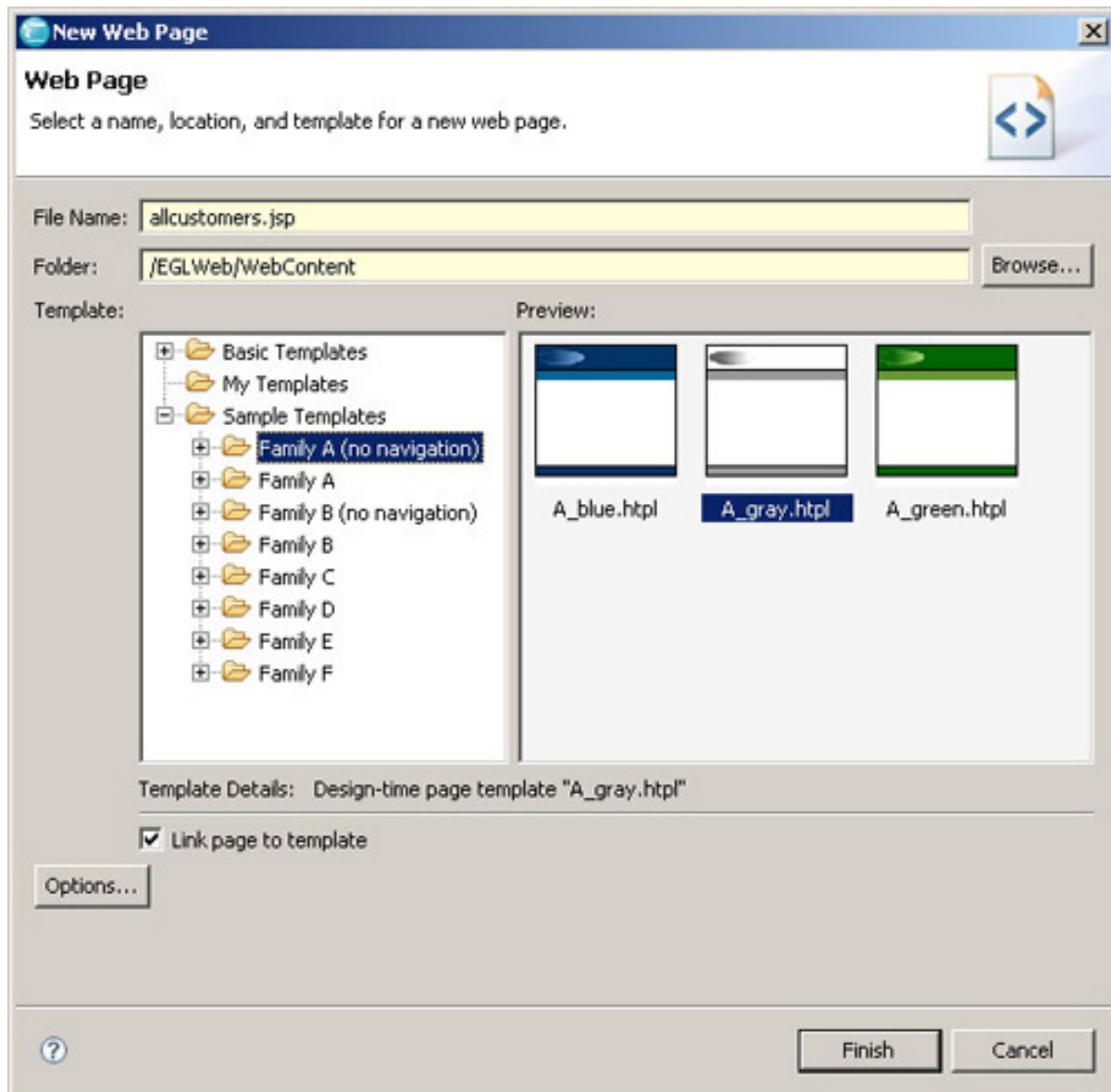
 [Show me](#)

1. In the Project Explorer view, click the WebContent folder in the EGLWeb project to select it. It is important to select the place where you want to put new files, or else the new files might not appear where you expect.

2. Click **File > New > Other**.
3. In the New window, expand **Web** and select **Web page**.
4. Click **Next**.
5. When the New Web Page window opens, in the File Name field, type `allcustomers.jsp` (be sure to include the extension).
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, select the **A_gray.html** template.

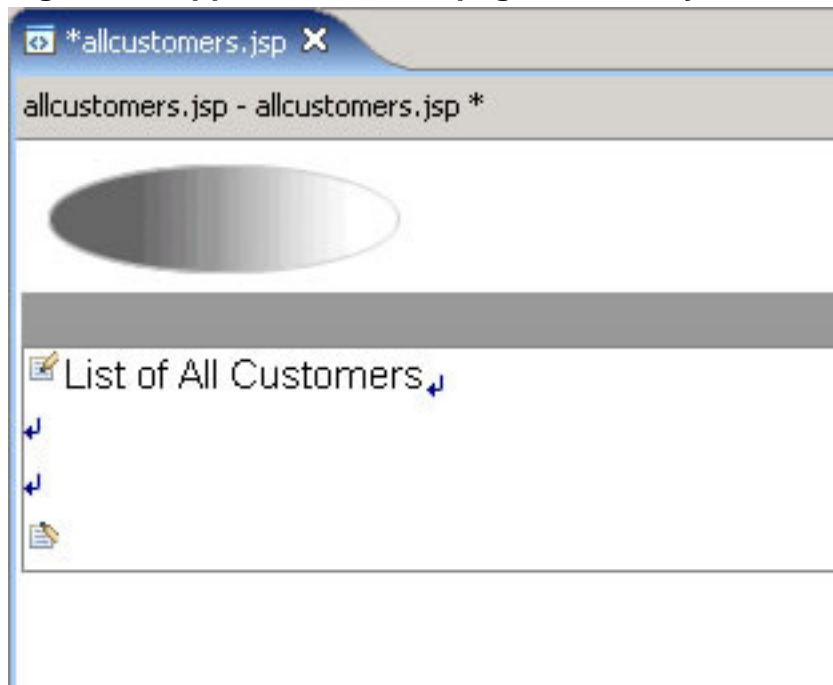
The New Web Page window should now look like Figure 16.

Figure 16. New Faces JSP file screen with the necessary fields filled in



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 section.

The page should now look like Figure 17.

Figure 17. Appearance of the page before any EGL content is added

13. Save the file.

Preview the Web page on the server

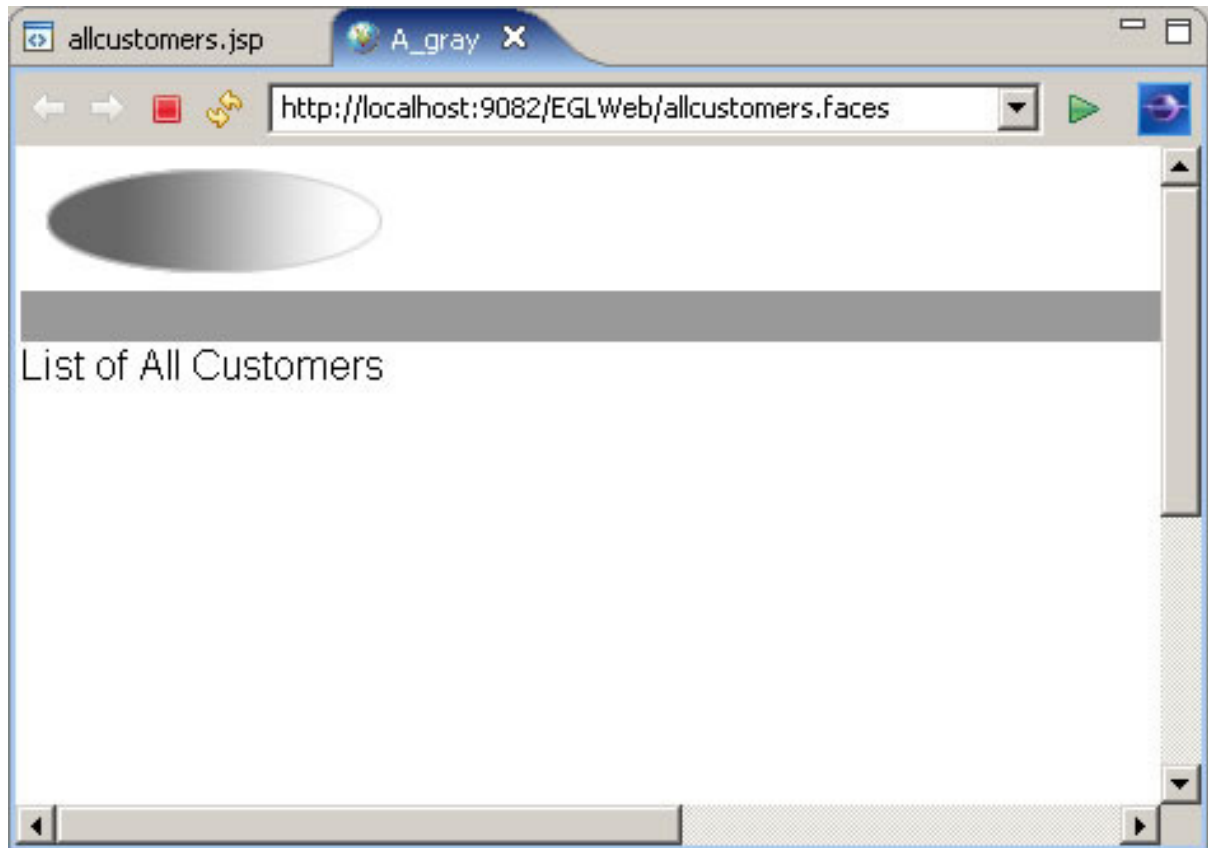
No data from the database is on the Web page yet, but you can run the JSP file on the Web application server and see what the page looks like so far.

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 drop-down menu, select **Run As > Run on Server**.
3. When the "Define a new server" window opens, select **Choose an existing server**.
4. Under "Select the server that you want to use," select **WebSphere Application Server v6.1**.
5. Select the **Set server as project default (do not ask again)** check box.
6. Click **Finish**.

The Web page opens in a Web browser inside the workbench. The page should now

look like Figure 18.

Figure 18. Appearance of the page in the internal Web browser



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 added yet. In the next section, you use EGL to add data.

Section 8. Add data to the page

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

This task includes these steps:

1. Add fields to the Page Data view and associate those fields with the Web page.
2. Place fields from the Page Data view on the related Web page.
3. 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. (In earlier 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

Would you like to see these steps demonstrated?



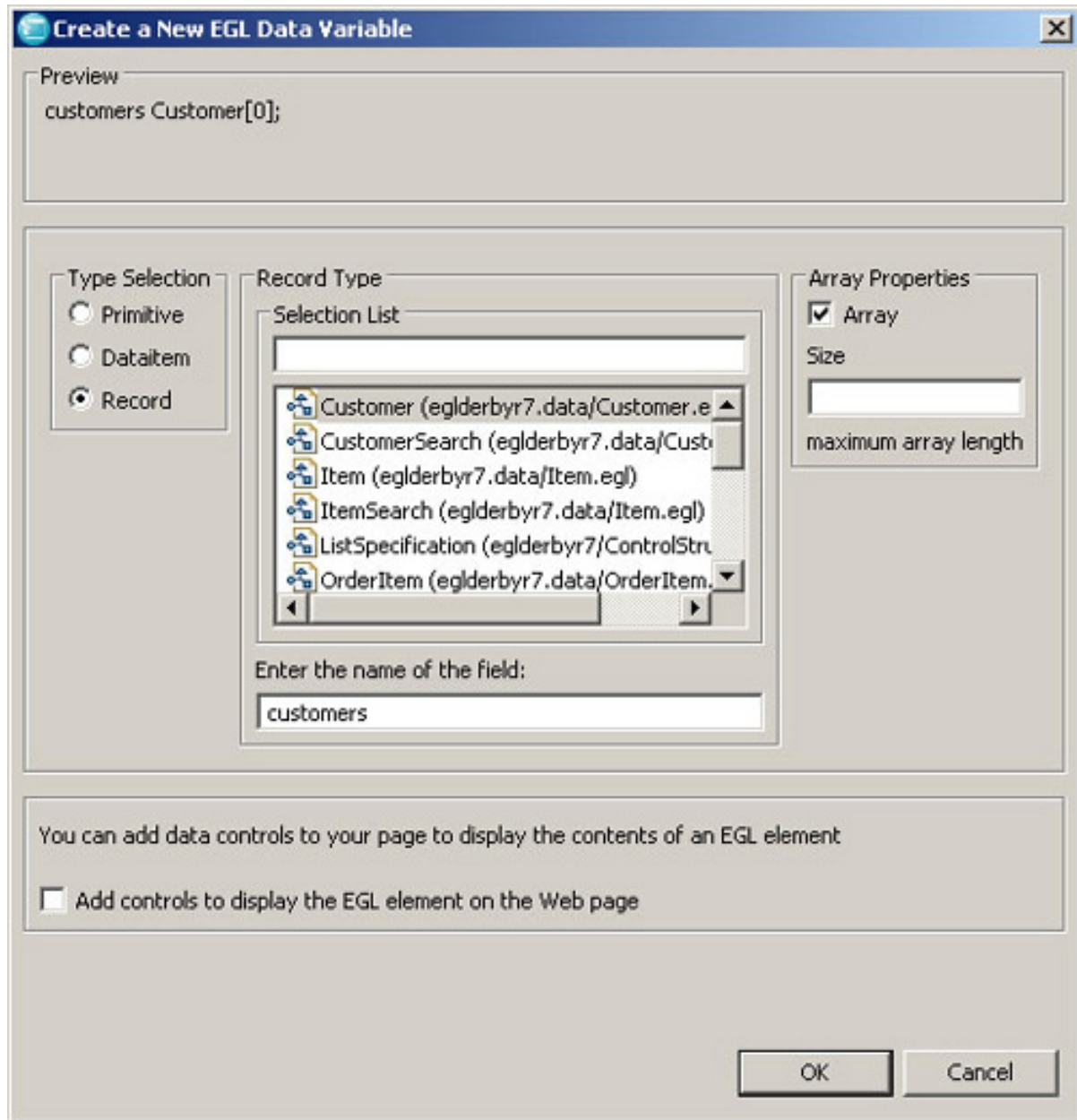
[Show me](#)

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, select **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, select **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.
6. When the Create a New EGL Data Variable window opens, under Type Selection, select **Record**.

7. Under Record Type, select **Customer**. In this way, you select the Record part on which each of the array elements will be based.
8. For "Enter the name of the field," type this text:
`customers`.
9. Under Array Properties, select the **Array** check box. Leave the Size field blank.
10. Clear the check box that says "Add controls to display the EGL element on the Web page."

The Create a New EGL Data Variable window should now look like Figure 19.

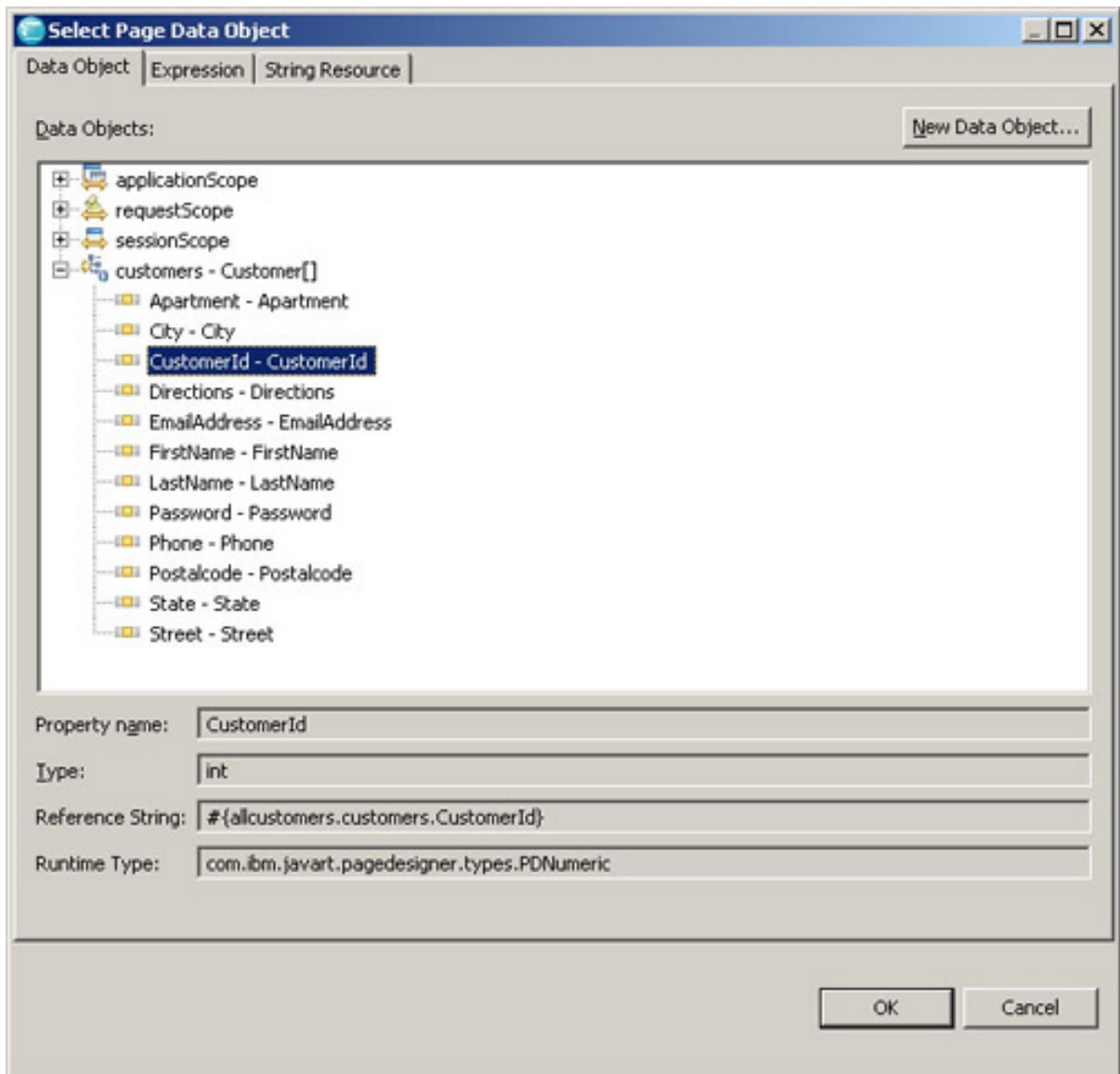
Figure 19. Create a New EGL Data Variable window with the necessary fields filled in



11. Click **OK**.
12. When an item that represents the new variable appears in the Page Data view under JSF Handler, expand **JSF Handler** and then expand **customers - Customer[]**. There are 12 icons beneath this icon, which represent the 12 fields in this database record.

The Page Data view should now look like Figure 20.

Figure 20. Page Data view with customers array variable visible



By adding entries to the Page Data view, you have also added an array of records to the JSF handler. Next, you 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 **allcustomers.jsp** file, releasing it below the "List of All Customers" text,

in the midst of the blank lines that you added in the previous exercise.

The Insert List Control window will open. This window lists all of the fields in the database record. You can use this window to choose the fields that will be shown on the page.

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

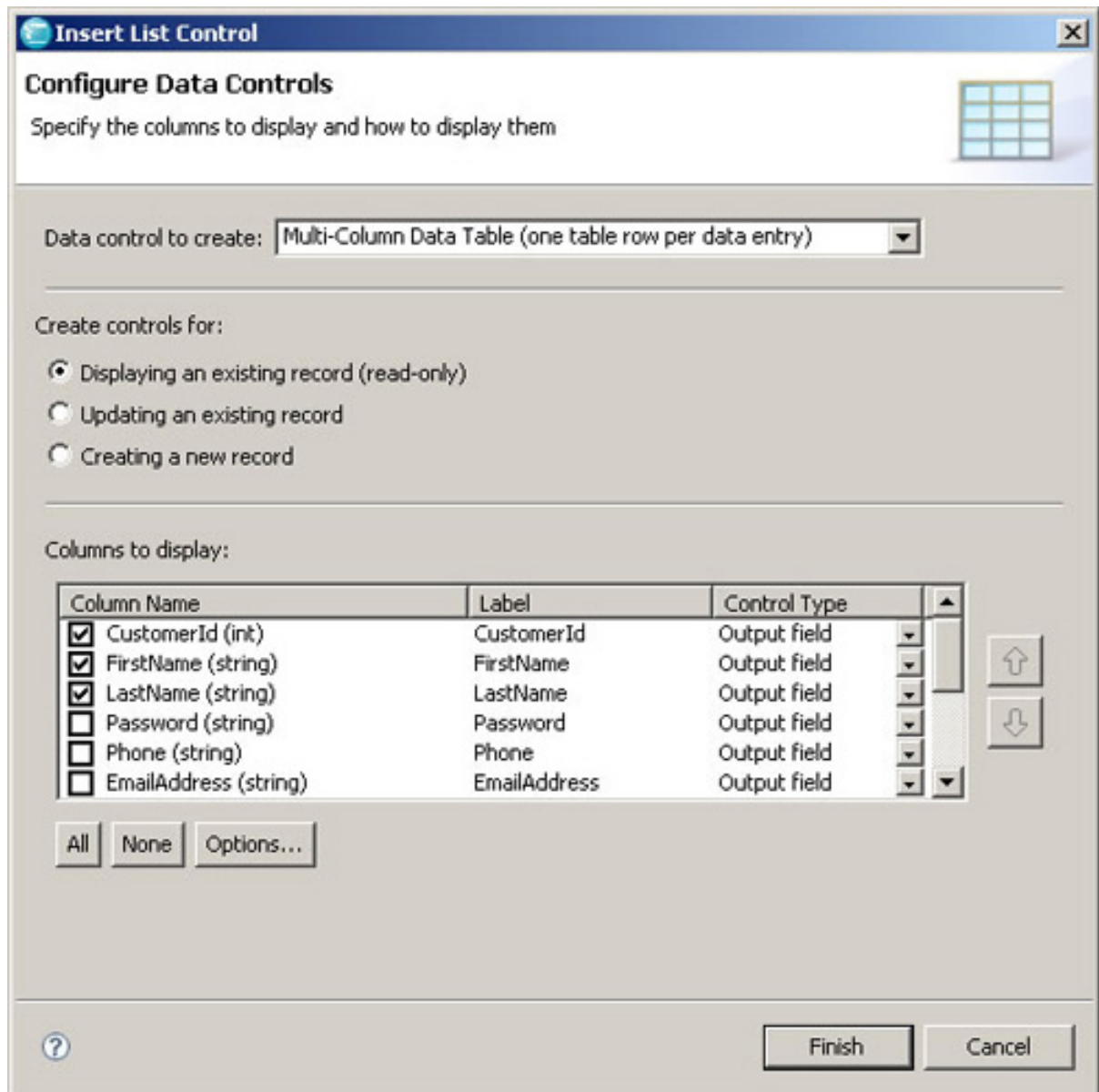
Note:

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 where you can bind actions. You'll create this type of field on another page. For the purpose of this tutorial, 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," select **None**. You have deselected all of the fields.
5. Select the check boxes next to these fields:
 - customerId
 - firstName
 - lastName

The Insert List Control window should look like Figure 21.

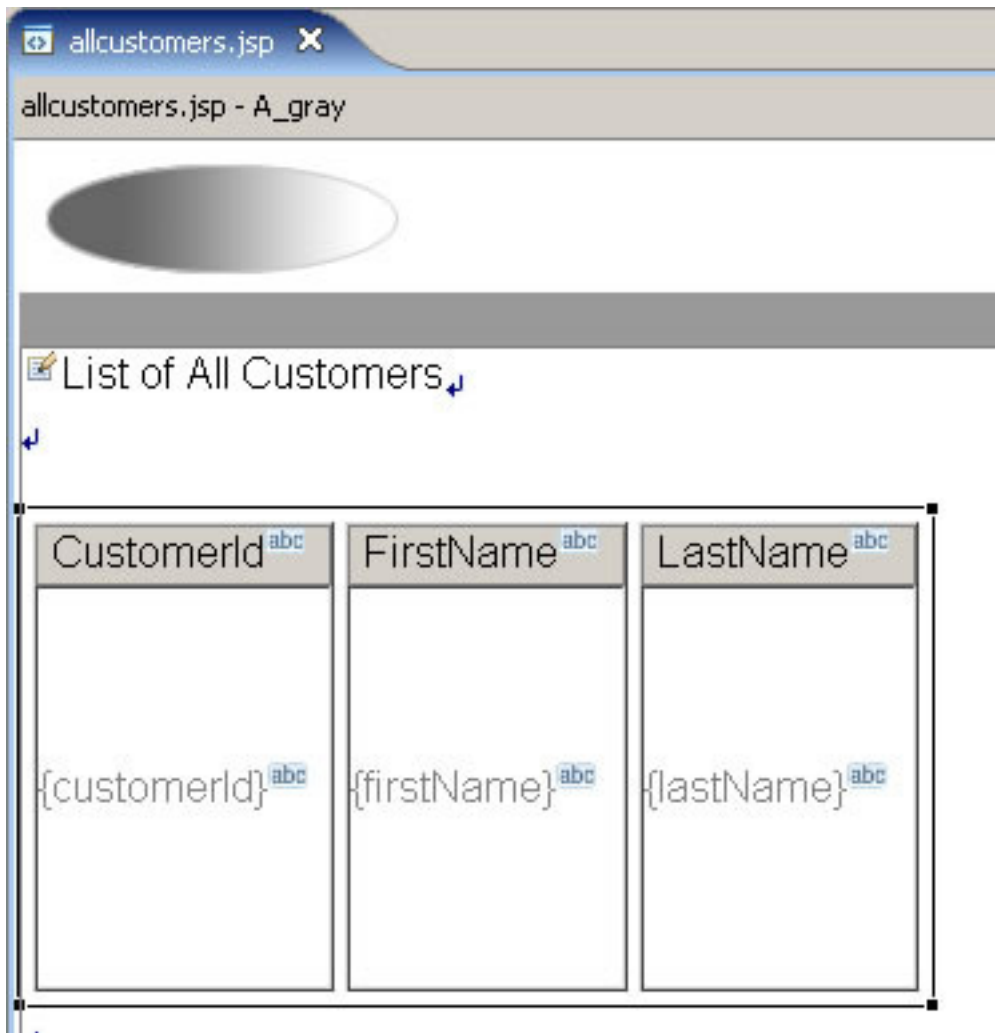
Figure 21. Insert List Control window with fields selected to be displayed on the page



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

The page should now look like Figure 22.

Figure 22. Appearance of the page with EGL data being shown



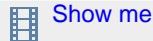
The columns in the data table have headings based on the names of the fields in the database. You can change these headings by clicking them, opening the **Properties** view, and changing the Value field.

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.

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.

Would you like to see these steps demonstrated?



1. Right-click anywhere in the free-form area of the **allcustomers.jsp** file.
2. From the drop-down 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 this line of code:

```
customers Customer[0];
```

This is the 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 shown above, add this code, exactly as written:

```
status StatusRec;
```

5. On a blank line immediately after the first line of code in the file, the line 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 previously.

6. In the **allcustomers.egl** file, add a blank line between these two 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 should look like the code in Listing 1.

Listing 1. Beginning of the JSF Handler part code

```
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 EGL part types can accept one or more properties, and each kind of part can accept different properties. In this case, the JSF Handler has these three properties defined:

- `onConstructionFunction = onConstruction`:
The `onConstructionFunction` property 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 `onPreRenderFunction` property specifies a function in the JSF Handler that runs each time that the page is shown in the browser. Unlike the `onConstructionFunction` property, the function defined in the `onPreRenderFunction` property runs each time that 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 that the page loads. The property specifies that the name of this function is `onPreRender()`, and EGL will report an error if there is not a function with this name in the part.
- `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.

8. Below the `status StatusRec;` code, add this 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 this code: `cust`
 - C. Press **Control** and the **space bar** simultaneously (Cntrl+space bar). The code completion window opens with all of the available EGL commands that begin with `cust`.
 - D. From the code completion window, select the `CustomerLib` library, either by highlighting it with the mouse cursor and then pressing Enter or by double-clicking it. Now the new line of code reads `CustomerLib`.
 - E. Type a period after `CustomerLib`.
 - F. Press **Control+space bar** again to open the code completion window 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 button. 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 `customers`, the name of your record variable.
 - I. End the line of code with a semicolon.

This is how the new line of code should read:

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

Also notice that there is a new `import` statement near the top of the file that reads:

```
import eglderbyr7.access.CustomerLib;
```

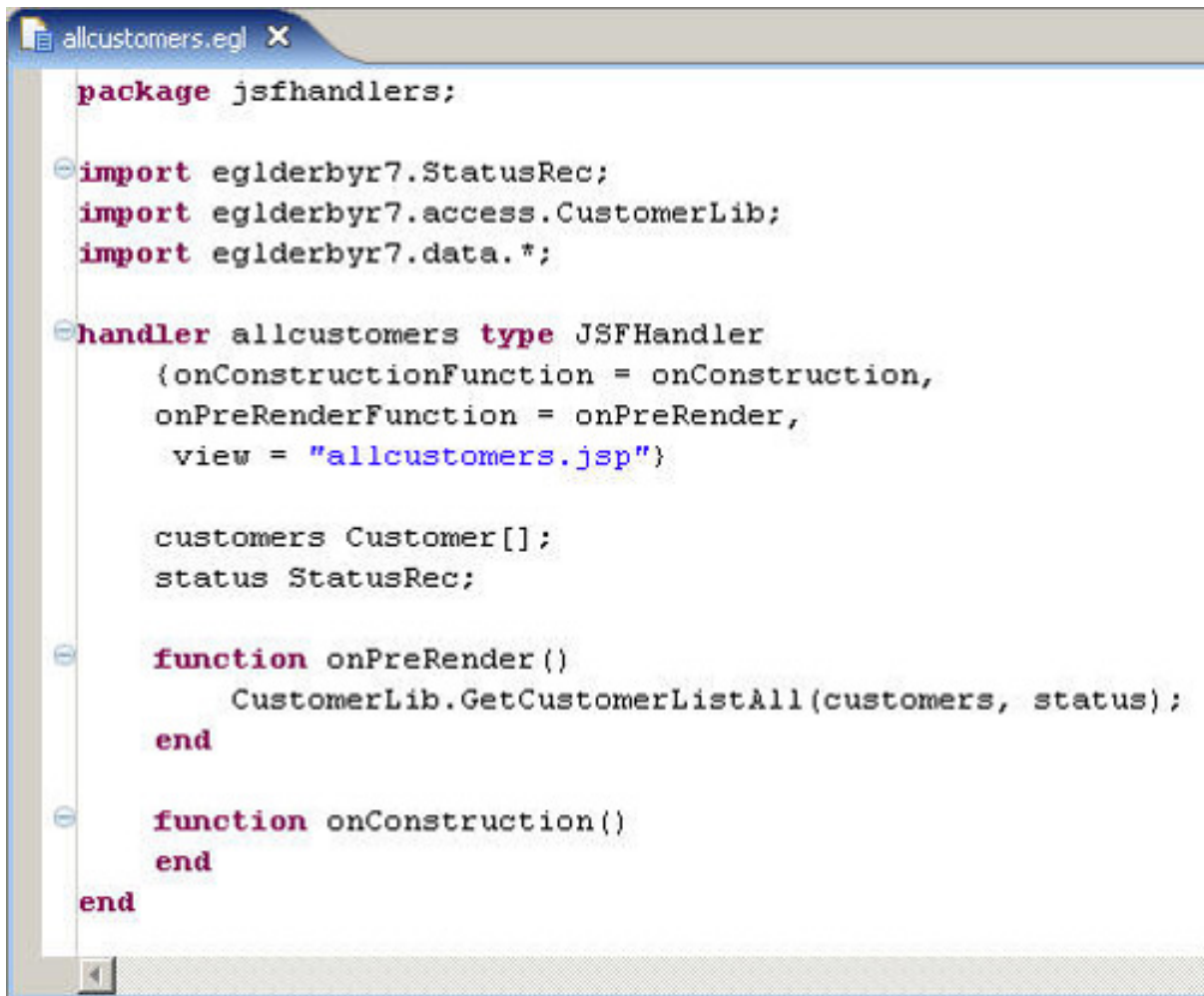
This line imports the library so that you do not need to write the complete path to the library in your code and, instead, can refer to it directly.

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

11. Be sure to save the file.

Now the `allcustomers.egl` file should look like Figure 23.

Figure 23. Code for `allcusomters.egl`



```
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[];
  status StatusRec;

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

  function onConstruction()
  end

end
```

This 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 [Appendix A](#).

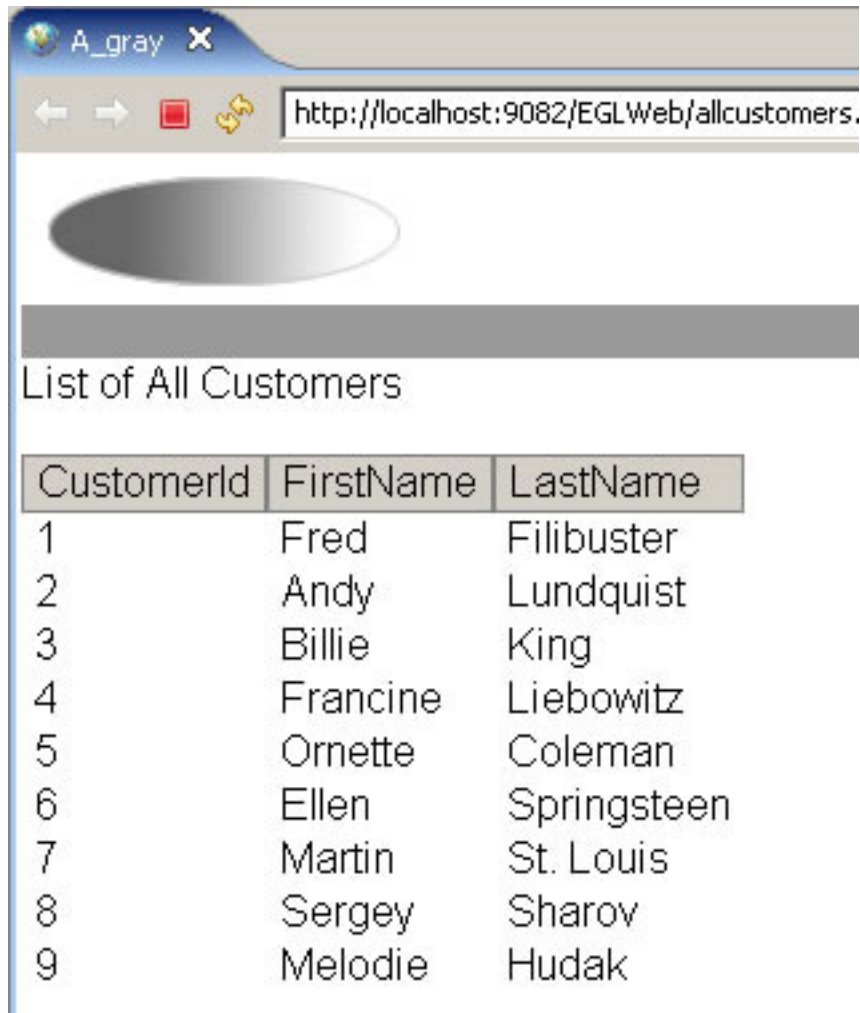
Test the page

Now the page is ready to be run on the server. To test the page and see how the database data appears on the page.

1. In the Project Explorer, right-click the EGLWeb project and then select **Generate**.
2. Still in the Project Explorer, right-click the **allcustomers.jsp** file, *not* the allcustomers.egl file.
3. From the drop-down menu, select **Run As > Run on Server**. As in the previous section, the Web page opens in a Web browser inside the

workbench. This time, the dynamic data appears on the page. The page should look like Figure 24.

Figure 24. The allcustomers.jsp page as shown in the internal Web browser




Section 9. Pass a parameter to another page

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

Add the link to the allcustomers.jsp file

Would you like to see these steps demonstrated?

 [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, select the **Link** component.
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 that you will create in the next section to show only one row in the database, but with a `faces` extension rather than a `jsp` extension.

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

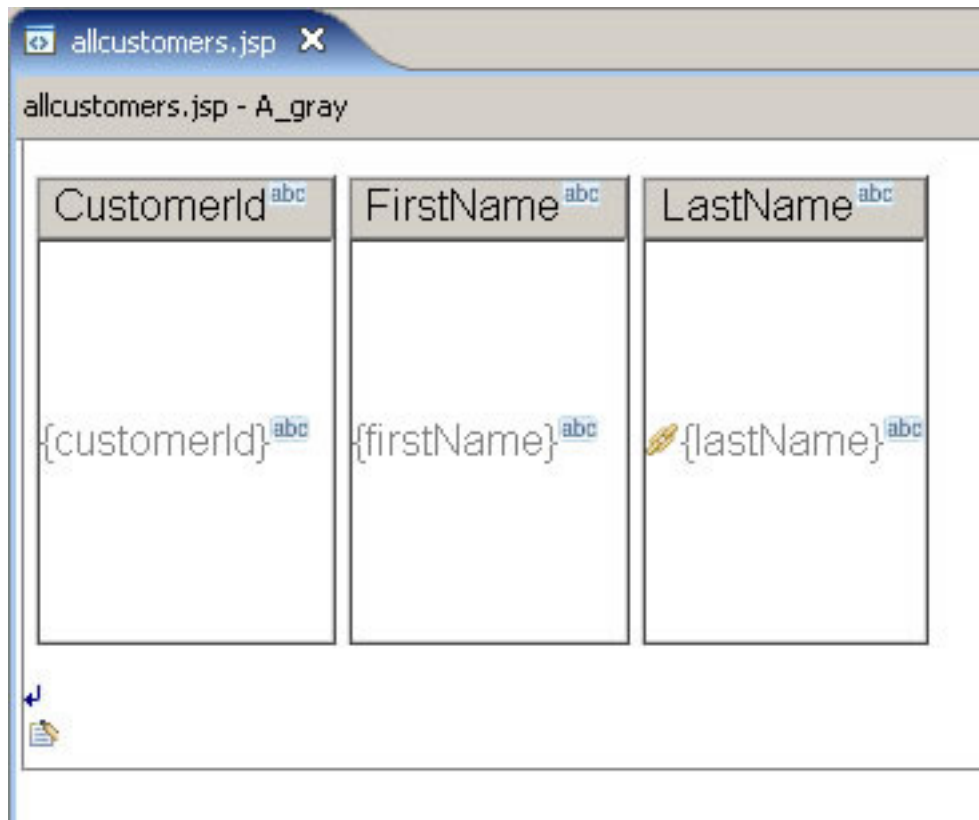
Tip:

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. Choose **Edit > Undo** and try again.

8. Save the page.

The page should now look like Figure 25.

Figure 25. allcustomers.jsp with a link to updatecustomer.jsp



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 need to specify an **HTTP request parameter** for the link that 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 that you just added to the `{lastName}` text field.

Tip:


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, which is usually at the bottom of the workbench. If you cannot find the Properties view, choose **Window > Show View >**

Properties.

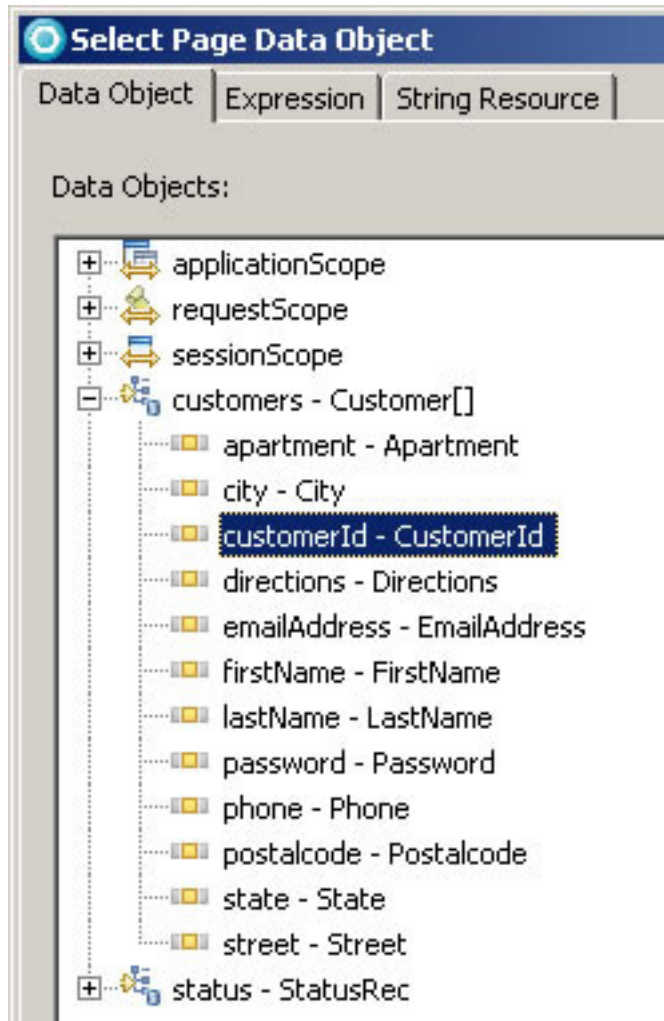
3. In the Properties view, select the **Parameter** tab, directly below the `hx:outputLinkEx` tab. If you can't find the Parameter tab, be sure that you have clicked directly on the icon to select it.
4. Click **Add Parameter**. A new parameter named `Name0` is added to the list of parameters.
5. Click the **Name** column and type this text as the new name of the parameter:

CID

6. Click the **Value** column.
7. Click the **Select Page Data Object** button () to open the Select Page Data Object window.
8. Under Data Objects, click the **+** (plus) symbol to expand **customers - Customer[]**.
9. Click **customerId**.

The Select Page Data Object window should now look like Figure 26.

Figure 26. Select Page Data Object window, showing the value of the parameter to be added to the link



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

The value of the link's `CID` parameter is now bound to the value of the `customer_id` field. When the user clicks on the link, the runtime code invokes the `updatecustomer.jsp` file and makes the customer ID number available to the `onPreRender()` function of the related JSF handler.


In the next section, you create the Web page for the `updatecustomer.jsp` file. Later, you set up the JSF handler to receive the parameter and to show only the customer with that ID number.

Section 10. Create an update page

Now you 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.

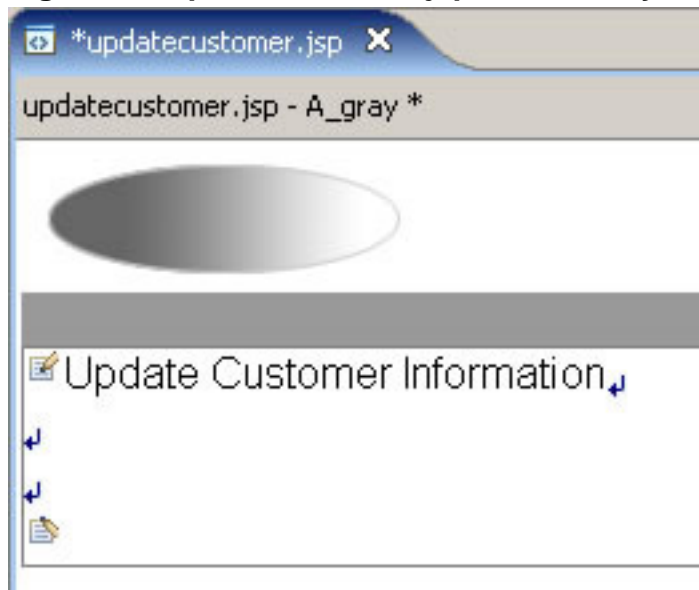
Create the updatecustomer.jsp file

Would you like to see these steps demonstrated?

 [Show me](#)

1. In the **Project Explorer** view, select the **WebContent** folder of the EGLWeb project.
2. Choose **File > New > Other**.
3. In the New window, expand **Web** and select **Web page**.
4. Click **Next** to open the New Web Page window.
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, select **My Templates**.
8. In the Preview box, select the **A_gray.html** template.
9. Click **Finish**.
10. After the new page is created, it opens in the editor. 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 Update Customer JSP should look like Figure 27.

Figure 27. updatecustomer.jsp without any EGL data

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 can 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** in the Palette view.
2. Drag the **New Variable** icon onto the page, below the text that says "Update Customer Information."
3. When the Create a New EGL Data Variable window opens, under Type Selection, select **Record**.
4. Under Record Type, select **Customer**.
5. For **Enter the name of the field** field, type

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, select **Updating an existing record**.
10. Click **Options** to open the Options window.
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:

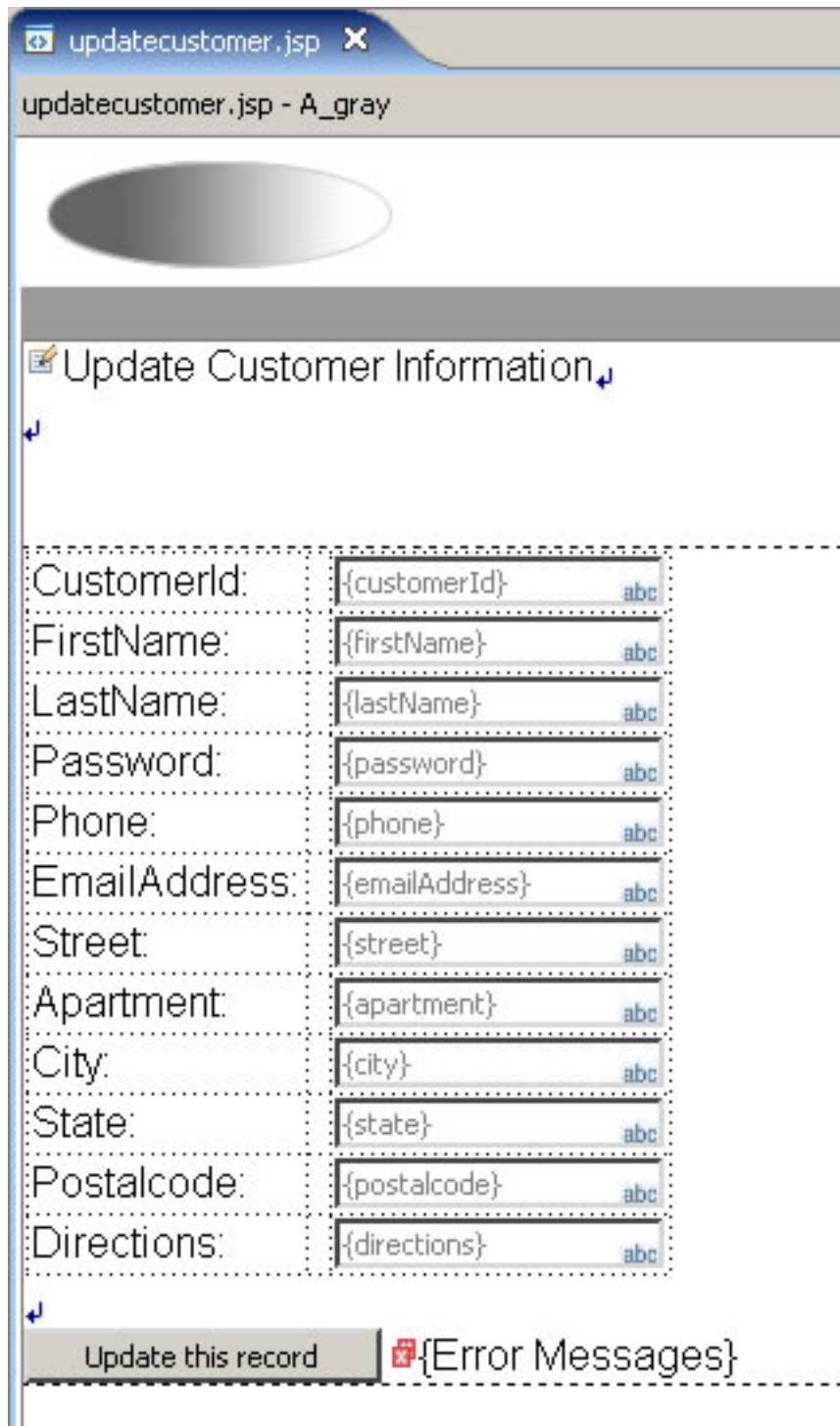
```
Update this record
```

14. Click **OK**.
15. Click **Finish**.
16. Save the page.

The data controls for updating the record are inserted on the Web page. Notice 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 runtime error messages will be displayed.

The page should now look like Figure 28.

Figure 28. updatecustomer.jsp with EGL data displayed



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 that you added a link to pass the

customer ID number in a parameter named `CID`. In these steps, you 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 `updatecustomer.jsp` file.
2. From the drop-down menu, select **Edit Page Code**. The `updatecustomer.egl` file opens in the editor.
3. In the `updatecustomer.egl` file, find this line:

```
customer  
Customer;
```

4. 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 `customer Customer;` line, add this code, exactly as written:

```
status StatusRec;
```

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

```
import eglderbyr7.StatusRec;
```

6. 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. Just as you did on the `allcustomers` page, add the `onPreRender` attribute to the Handler, so that the beginning of the Handler definition looks like Listing 2.

Listing 2. Code to add the `onPreRender` attribute to the JSF Handler

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

7. Then, just as you did in the `allcustomers` page, add the `onPreRender()` function:

```
function onPreRender()  
end
```

8. Change the line that says `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 `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. Tip: You can use the code completion feature that you learned about in [Add data to the page](#).

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

The `GetCustomer()` function works just like the `GetCustomerAll()` function that you used previously, but `GetCustomer()` retrieves one record, whereas `GetCustomerAll()` 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
```

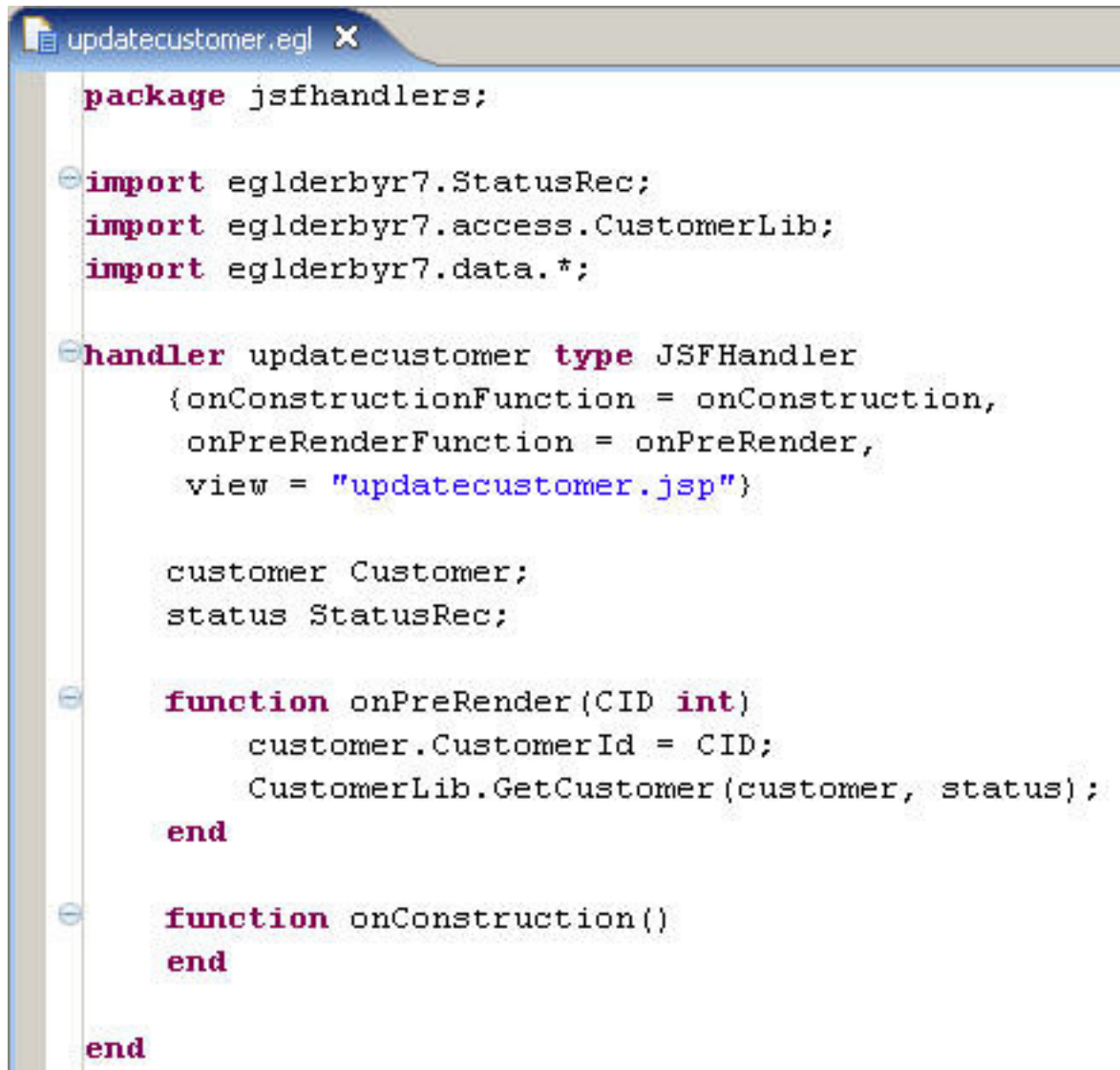
11. If you did not use code completion to add the `GetCustomer` function, add the following `import` statement with the other `import` statement, immediately above the line that begins handler `updatecustomer`:

```
import eglderbyr7.access.CustomerLib;
```

12. Save the file.

The JSF handler should now look like Figure 29.

Figure 29. The updatecustomer.egl file configured to receive a parameter and display the specified record



```
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. Currently, you can change the information in the fields on the Web page, but there is no function to send those updates to the database. Next, use the `UpdateCustomer()` function to make those updates to the database.

Update the record in the database

Here, you add a new EGL function in the JSF handler named `updateRecord`. Then

you bind this function to the button that 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 that 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 can use code completion or copy the function from this page to make sure that 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` file in the editor. You may still have this page open in the editor tabs. If you can't find it there, in the Project Explorer view, in the `EGLWeb/WebContent/allcustomers.jsp` folder, double-click the `updatecustomer.jsp` file.
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 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.

Figure 30 illustrates the complete code of the `updatecustomer.egl` file. If you see any errors marked by red **X** symbols in the file, make sure that your code matches the code in [Appendix B](#).

Figure 30. The `updatecustomer.egl` file, including a function to update a specified record

```
updatecustomer.ejl 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

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

end
```

Test the finished site

The site is now ready to test. You can 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 select **Generate**.
2. Still in the Project Explorer view, right-click the **allcustomers.jsp** file and then select **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. That will take you 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.

You should have returned to the allcustomers.jsp file. Notice that the record has changed to show the new first name that you typed. You can click that record's last name again to see the new information that was saved in the database.

Section 11. Lessons learned

By completing this tutorial, you have learned how to use Rational Business Developer to complete these 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 to delete 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 to create a new customer record. You will need to create a new Web page similar to the `updatecustomer.jsp` page that uses the `createCustomer()` function.

Section 12. Appendix A. `allcustomers.egl` after the section Add data to the page

The code in Listing 3, which follows, is the completed version of the `allcustomers.egl` file. If you see any errors marked by red **X** symbols in the file, make sure that your code matches this code:

Listing 3. Completed code for the `allcustomers.egl` file

```
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[];
  status StatusRec;

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

  Function onConstruction()
  End

End
```

Return to [Add data to the page](#).

Section 13. Appendix B: The `updatecustomer.egl` file after the section Create an update page

The code that follows in Listing 4 is the completed version of the updatecustomer.egl file. If you see any errors marked by red X symbols in the file, make sure that your code matches this code:

Listing 4. Completed code for the updatecustomer.egl file

```
package jsfhandlers;

import eglderbyr7.egl.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 [Create an update page.](#)

Downloads

Description	Name	Size	Download method
Sample Derby database used in the tutorial	EGLDerbyR7.zip	116KB	HTTP

[Information about download methods](#)

Resources

Learn

- Learn more about IBM Rational Business Developer and the IBM Enterprise Generation Language (EGL):
 - Attend a free technical briefing about [Rational Business Developer and the power of the Enterprise Generation Language \(EGL\)](#). Registration required.
 - Read the [Enterprise Generation Language executive overview](#) on developerWorks (March 2008). Learn why IBM developed EGL and how IT organizations can use it to become more productive, more quickly in today's services-based arena.
- Learn more about the Apache Derby open source database:
 - Browse the [Apache Derby project resources](#) that are available on IBM developerWorks.
 - Visit the [Apache Derby Web site](#).
- Check the [Rational software area on developerWorks](#) for technical resources and best practices for Rational Software Delivery Platform products.
- Explore [Rational computer-based, Web-based, and instructor-led online courses](#). Hone your skills and learn more about Rational tools with these courses, which range from introductory to advanced. The courses on this catalog are available for purchase through computer-based training or Web-based training. Additionally, some "Getting Started" courses are available free of charge.
- Subscribe to the [Rational Edge newsletter](#) for articles on the concepts behind effective software development.
- Subscribe to the [IBM developerWorks newsletter](#), a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download the free trial versions of the software required for this tutorial:
 - [IBM Rational Business Developer](#).
 - [IBM Rational Application Developer for WebSphere Software V7.0](#), which is a prerequisite for Rational Business Developer. It helps developers

quickly design, develop, analyze, test, profile, and deploy high-quality Web, SOA, Java, J2EE, and portal applications.

- Download [trial versions of other IBM Rational software](#).
- Download these [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Tivoli®, and WebSphere®.

Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).
- [Enterprise Generation Language \(EGL\) and Rational Business Developer forum](#): Ask questions about Rational Business Developer and EGL.

About the author

Tim McMackin



Tim McMackin is a technical writer for IBM's Enterprise Generation Language in Raleigh, NC. He has a background in writing for advertising technical products and has been with IBM since 2004.

Trademarks

IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States, other countries or both.

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

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