# Developing JSF 2.x Facelets

JavaServer Faces (JSF ) 2.x has introduced several new features, such as integrated

Facelets, implicit navigation, conditional navigation, preemptive navigation, bean

validation, view parameters, client behaviors, new scopes (view, flash, and custom),

configuration annotations, composite components, Resource Handler API, support

for Ajax, and new event handling and exception handling features.

WildFly 8.x supports JSF 2.2, the latest version of JSF. JSF 2.2 has introduced new

features of convenient HTML5 markup, Resource Library Contracts, Face Flows,

and stateless views. JSF 2.2 support is added to a project with a Maven dependency.

In this chapter, we will develop a JSF 2.x Facelets application in Eclipse, build and

package the application with Maven, and deploy the application to WildFly 8.1. We

will run the application in WildFly 8.1 to demonstrate the use of Facelets in a web

application. Facelets is the default **View Declaration Language** (**VDL**) in JSF 2.x,

replacing JSP as the default VDL. This Modulehas the following sections:

* Setting the environment
* Creating a Java EE web project
* Creating a managed bean
* Creating the Facelets template
* Creating a header and footer
* Creating input and output Facelets pages
* Creating a web descriptor
* Installing the web project with Maven
* Running the Facelets application

**[ 81 ]**

*Developing JSF 2.x Facelets*

## Setting up the environment

##### We need to install the following software:

* **WildFly 8.1.0.Final**: Download wildfly-8.1.0.Final.zip from

<http://wildfly.org/downloads/>

* **MySQL 5.6 Database-Community Edition**: Download this edition from [http://dev.mysql.com/downloads/mysql/.](http://dev.mysql.com/downloads/mysql/) When installing MySQL, also install **Connector/J**
* **Eclipse IDE for Java EE Developers**: Download Eclipse Luna from

[https://www.eclipse.org/downloads/packages/release/Luna/SR1](http://www.eclipse.org/downloads/packages/release/Luna/SR1)

* **JBoss Tools (Luna) 4.2.0.Final**: Install this as a plugin to Eclipse from the Eclipse Marketplace (<http://tools.jboss.org/downloads/> installation.html)
* **Apache Maven**: Download version 3.05 or higher from

<http://maven.apache.org/download.cgi>

* **Java 7**: Download Java 7 from <http://www.oracle.com/technetwork/>

java/javase/downloads/index.html?ssSourceSiteId=ocomcn

Set the environment variables JAVA\_HOME, JBOSS\_HOME, MAVEN\_HOME, and MYSQL\_ HOME. Add %JAVA\_HOME%/bin, %MAVEN\_HOME%/bin, %JBOSS\_HOME%/bin, and %MYSQL\_ HOME%/bin to the PATH environment variable.

Create a WildFly 8.1.0 runtime as discussed in *Module1*, *Getting Started with EJB 3.x*. Create a MySQL database CATALOG with the following SQL script:

CREATE TABLE CATALOG(CatalogId INTEGER

PRIMARY KEY, Journal VARCHAR(25), Publisher VARCHAR(25), Edition VARCHAR(25), Title Varchar(45), Author Varchar(25));

INSERT INTO CATALOG VALUES('1', 'Oracle Magazine', 'Oracle Publishing', 'Nov-Dec 2004', 'Database Resource Manager', 'Kimberly Floss');

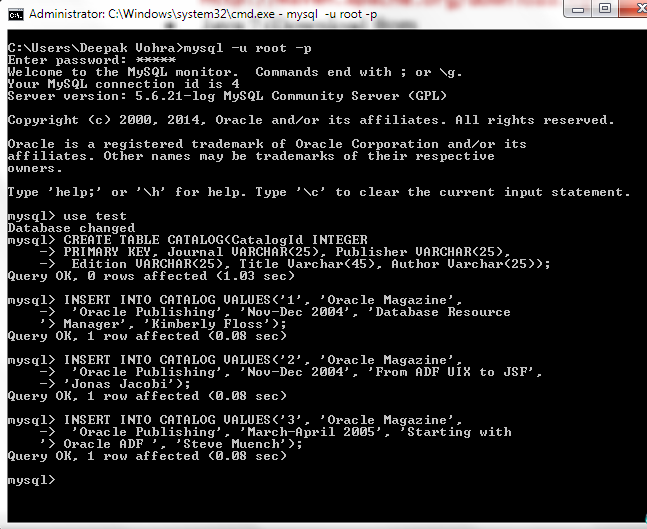
INSERT INTO CATALOG VALUES('2', 'Oracle Magazine', 'Oracle Publishing', 'Nov-Dec 2004', 'From ADF UIX to JSF', 'Jonas Jacobi'); INSERT INTO CATALOG VALUES('3', 'Oracle Magazine', 'Oracle Publishing', 'March-April 2005', 'Starting with Oracle ADF ', 'Steve

Muench');

**[ 82 ]**

*Module3*

##### Run the script in the MySQL 5.6 command-line client. The database table, Catalog, gets created. The output from the preceding script is shown in the following screenshot:



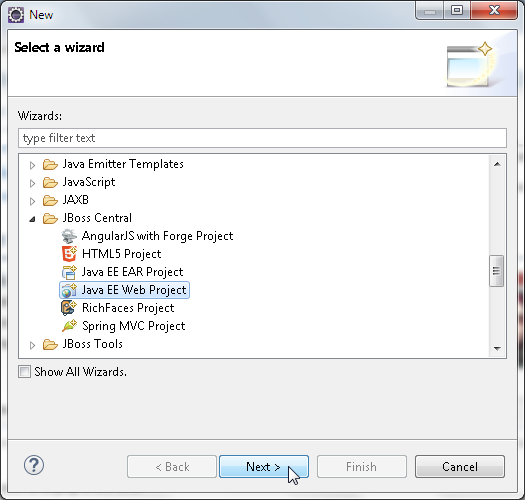
We also need to configure a data source for MySQL database. The procedure to configure a MySQL data source was discussed in *Module1*, *Getting Started with EJB 3.x*, and will not be repeated in this chapter.

**[ 83 ]**

*Developing JSF 2.x Facelets*

## Creating a Java EE web project

In this section, we will create a Eclipse project for JSF 2.x. Select **File** | **New** | **Other** in Eclipse. Select **JBoss Central** | **Java EE Web Project** and click on **Next**, as shown in the following screenshot:

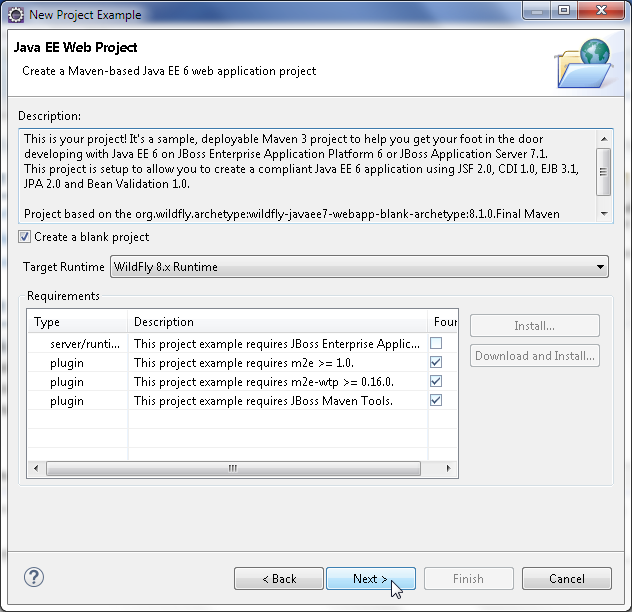


##### The **Java EE Web Project** wizard gets started. Though the wizard indicates that a Java EE 6 web application project will be created, a Java EE 7 web application project gets created.

**[ 84 ]**

*Module3*

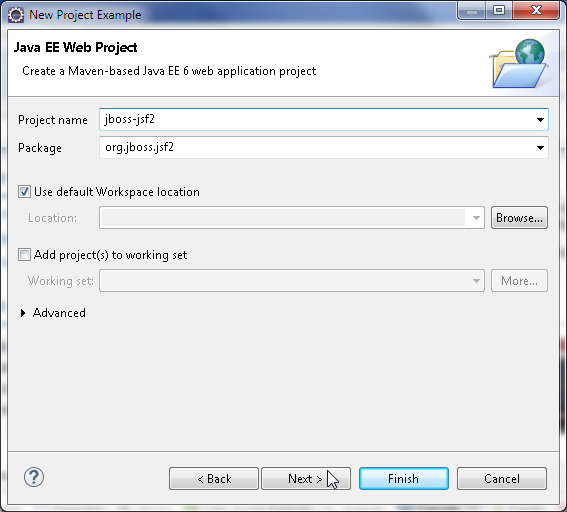
A test is run for the requirements, which include JBoss AS runtime, the **m2e** and **m2eclipse-wtp** plugins, and the JBoss Tools plugin, as shown in the following screenshot. Select **Create a blank project** checkbox and select **Target Runtime** as **WildFly 8.x Runtime**. Click on **Next**.



**[ 85 ]**

*Developing JSF 2.x Facelets*

Specify **Project name** (jboss-jsf2) and **Package** (org.jboss.jsf2), and select the checkbox **Use default Workspace location**. Click on **Next** as shown in the following screenshot:



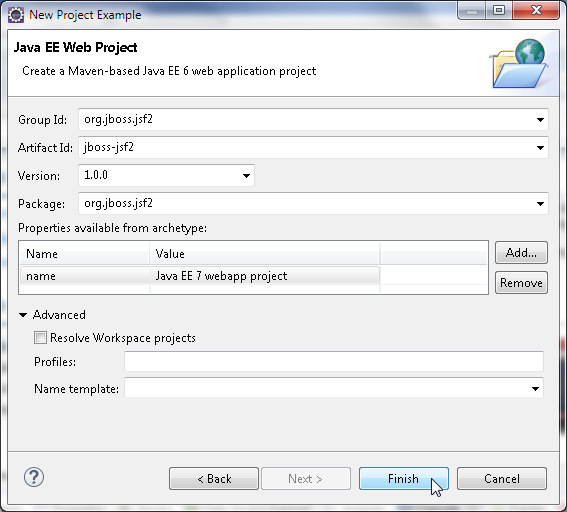
The Maven building tool is used for the example project, and therefore, it is necessary to specify the Maven modules: **Group Id** (org.jboss.jsf2), **Artifact Id** (jboss-jsf2), **Version** (1.0.0), and **Package** (org.jboss.jsf2), as shown in the following screenshot. Other than the name property, the properties listed in **Properties available from archetype** may be deleted as these are not required for the application.

**[ 86 ]**

*Chapter 3*

The default name property is set to **Java EE 6 webapp project**, which should be

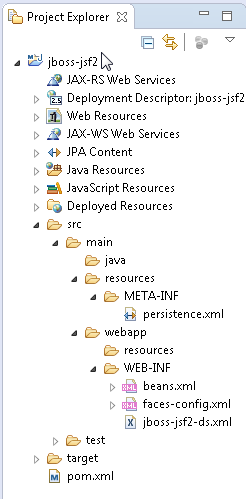
modified to **Java EE 7 webapp project**. Click on **Finish**.



**[ 87 ]**

*Developing JSF 2.x Facelets*

A **Java EE Web Project** gets created, as shown in **Project Explorer** in the following screenshot. Delete the //jboss-jsf2/src/main/resources/META-INF/ persistence.xml configuration file as it is not used in the JSF application. The web project includes a default.xhtml file in the WEB-INF/templates directory. We will need a Facelets template, but not the default template.



Facelets is a set of tags in the <http://java.sun.com/jsf/facelets>namespace. Facelets tags are used in conjunction with JSF Core and JSF HTML tag libraries to develop a JSF Facelets application. The default suffix for a Facelets page is .xhtml. A Facelets application consists of the following configuration and templating files:

1. **A Facelets template page**: A template may be reused in several Facelets composition pages.
2. **Facelets header and footer pages**: These pages are included in the Facelets template page for common sections of a Facelets application.
3. **A configuration file**: This is the faces-config.xml file, which is included by default in a Java EE web project.
4. **Facelets composition page or pages**: These pages are run on the WildFly.
5. **A managed bean**: This is used for the Facelets composition pages.

**[ 88 ]**

*Chapter 3*

We will create a Facelets application with an input Facelets composition page and an output Facelets composition page. A common header and footer are included in the input and output pages. In the input page, a SQL query may be specified in an input field. The SQL query is used to get data from the database and create a JSF data table and demonstrate templating. A command button sends the input request parameters to a managed bean's action method. In the action method, a connection is established with MySQL database and a result set generated for the SQL query. A JSF data table is generated from the result set and displayed in the output Facelets composition page.

In the subsequent sections, we will create the different components of the Facelets

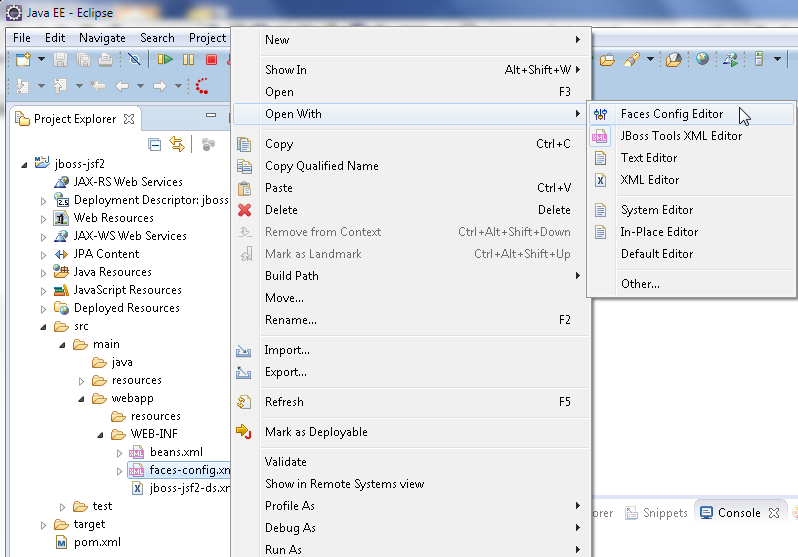
application; first, the managed bean.

## Creating a managed bean

A managed bean is a Java bean managed by the **JSF Managed Bean Facility**.

##### A managed bean is registered with JSF and instantiated when first invoked.

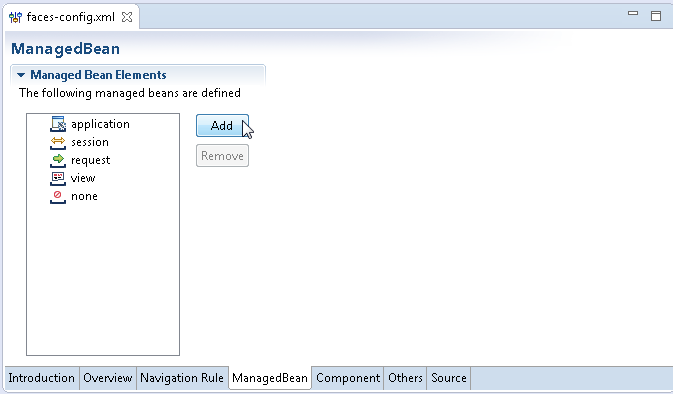
In this section, we will create a JSF managed bean for the Facelets composition pages. Right-click on faces-config.xml and select **Open With** | **Faces Config Editor**, as shown in the following screenshot:



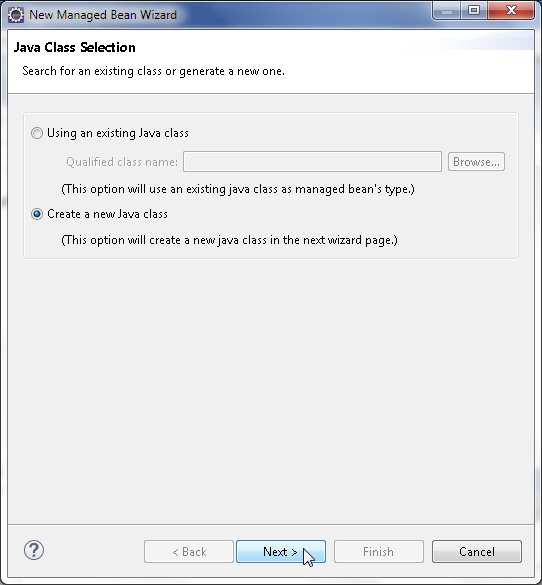
**[ 89 ]**

*Developing JSF 2.x Facelets*

The **Faces Config Editor** gets started. Click on **Add** to add a managed bean, as shown in the following screenshot:



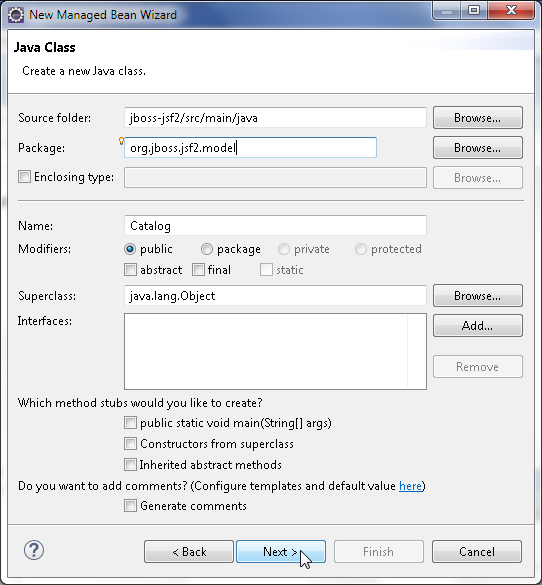
In the **New Managed Bean Wizard**, select **Create a new Java class** in **Java Class Selection** and click on **Next**, as shown in the following screenshot:



**[ 90 ]**

*Chapter 3*

The **Java Class** wizard gets started. The **Source Folder** (jboss-jsf2/src/main/ java), **Package** (org.jboss.jsf2.model), and **Name** (Catalog) should be specified, as shown in the following screenshot. Click on **Next**.

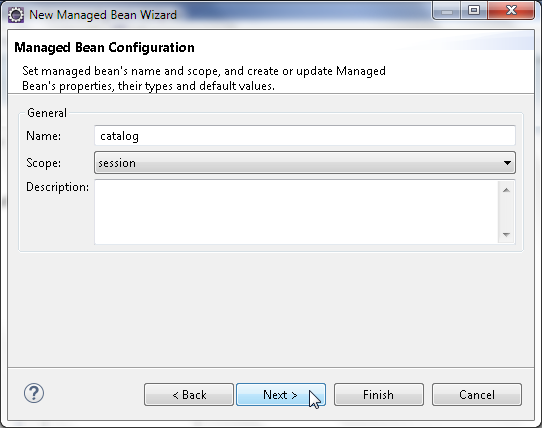


**[ 91 ]**

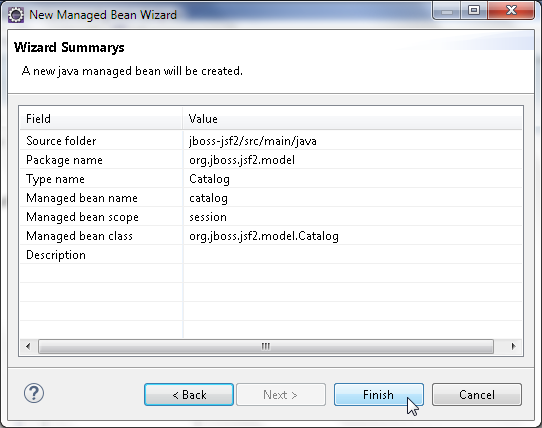
*Developing JSF 2.x Facelets*

In the **Managed Bean Configuration** window, the **Name** textbox is specified as

catalog, and **Scope** as session. Click on **Next**, as shown in the following screenshot:



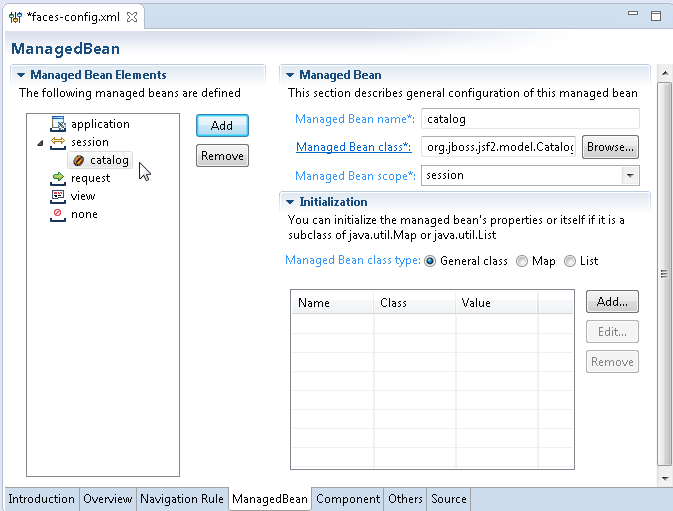
##### A summary of the managed bean to be created gets displayed in the **Wizards Summarys** window. Click on **Finish**, as shown in the following screenshot:



**[ 92 ]**

*Chapter 3*

##### A new managed bean catalog gets created, as shown in the following screenshot:



In the managed bean, import the required classes and annotate the Catalog class with the @ManagedBean annotation. Declare Java bean properties for an input form (of the type HtmlForm), input the text field (of the type HtmlInputText), a label for the input text field (of the type HtmlOutputLabel), an command button (of the type HtmlCommandButton), a data table (of the type HtmlDataTable), the

##### columns of the data table (of the type UIColumn), and an error message (of the type HtmlOutputText). Add getter/setter methods for the properties as required by the Java bean convention. Also, declare variables for a Connection object, a Statement object, and a ResultSet object.

**[ 93 ]**

*Developing JSF 2.x Facelets*

##### Add an action method-a method that has no parameters and returns a String value and commandButton1\_action. In the action method, create a connection to MySQL database either with the JDBC API or using a MySQL data source. If the JDBC API is used, the Connection object is obtained as follows:

Class.forName("com.mysql.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/test";

Connection connection = DriverManager.getConnection(url, "root", "mysql");

##### If a MySQL data source is used, the Connection object is obtained as follows:

InitialContext initialContext = new InitialContext(); DataSource ds = (DataSource) initialContext.lookup("java:jboss/ datasources/MySQLDS");

java.sql.Connection conn = ds.getConnection();

Create a Statement object from the Connection object using the createStatement method. Run the SQL query input in the input field using the executeQuery method to obtain a ResultSet:

Statement stmt = connection.createStatement(ResultSet.TYPE\_SCROLL\_ INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

ResultSet rs = stmt.executeQuery((String) inputText1.getValue());

Create a HtmlDataTable object, set its border using the setBorder method, set the cell padding using the setCellpadding method, and set the iteration variable for a data collection using the setVar method:

HtmlDataTable dataTable1 = new HtmlDataTable(); dataTable1.setBorder(5); dataTable1.setCellpadding("1"); dataTable1.setVar("journalcatalog");

Create a ResultSetDataModel object, which encapsulates a data collection represented by a ResultSet. Set the ResultSet generated from the SQL query as the data collection for the ResultSetDataModel object using the setWrappedData method:

ResultSetDataModel dataModel = new ResultSetDataModel(); dataModel.setWrappedData(rs);

**[ 94 ]**

*Chapter 3*

Create columns for the data table using UIColumn class constructors, and set the columns on the data table using the setColumn method of the HtmlDataTable object. Create a data table column header using the HtmlOutputText type variable for each column, and set a header on a column using the setHeader method of the UIColumn object. The data table values are also of the type HtmlOutputText. The result set data is bound to the data table using value expressions. A ValueExpression object is used to set values. An ExpressionFactory object is required to create a ValueExpression object. Obtain a ExpressionFactory object from a FacesContext object using getApplication().getExpressionFactory(). First create a FacesContext object using the getCurrentInstance() method. Create a ValueExpression object from the ExpressionFactory method using the createValueExpression(ELContext

context, java.lang.String expression, java.lang.Class<?> expectedType) method. The ELContext object for the createValueExpression method is created from the FacesContext object using the getELContext() method. The expression

for the createValueExpression method is an EL expression, and the expectedType method is String.class. A ValueExpression value is set on a HtmlOutputText type column using the setValueExpression method. Add the HtmlOutputText object to

##### a UIColumn object using getChildren().add(). The result set values are not bound on each data table cell individually, but an EL expression consisting of an iteration variable is used to bind the result set. For example, column1, which is the column for the catalog ID, is set as follows:

HtmlOutputText column1Text = new HtmlOutputText(); FacesContext fCtx = FacesContext.getCurrentInstance(); ELContext elCtx = fCtx.getELContext();

ExpressionFactory ef = fCtx.getApplication().getExpressionFactory(); ValueExpression ve = ef.createValueExpression(elCtx,"#{journalcatalog. catalogid}", String.class);

column1Text.setValueExpression("value", ve); column1.getChildren().add(column1Text);

Similarly, set the other data table columns. Set the ResultSetDataModel data collection on the HtmlDataTable object using the setValue() method:

dataTable1.setValue(dataModel);

##### If an error is generated, return the error from the action method, which navigates the Facelets application to error.jsp using implicit navigation. If implicit navigation is used, the to-URL is the same name as the String value returned by the action method.

For a more detailed discussion on implicit navigation and other JSF 2 features, refer to *JavaServer Faces 2.0, Essential Guide for Developers*, *Cengage Learning*. If an error is not generated, return the output that navigates to output.jsp to display the data table. The Catalog.java managed bean is available in this chapter download.

**[ 95 ]**

*Developing JSF 2.x Facelets*

##### As we have used the @ManagedBean annotation, the faces-config.xml file is an empty file, which is an advantage in terms of having to specify less configuration:

<?xml version="1.0" encoding="UTF-8"?>

<!-- This file is not required if you don't need any extra configuration. -->

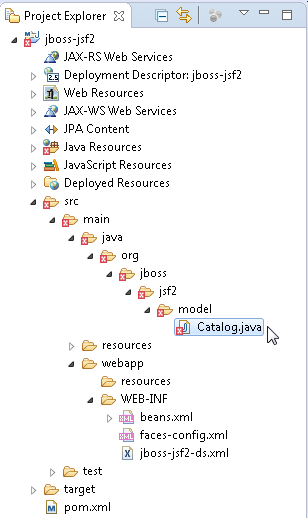
<faces-config version="2.0" [xmlns="http://java.sun.com/xml/ns/javaee"](http://java.sun.com/xml/ns/javaee) [xmlns:xi="http://www.w3.org/2001/XInclude"](http://www.w3.org/2001/XInclude) [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xsi:schemaLocation=" <http://java.sun.com/xml/ns/javaee>

[http://java.sun.com/xml/ns/javaee/web-facesconfig\_2\_0.xsd">](http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd)

</faces-config>

##### The managed bean is shown in the jboss-jsf2 project in the next screenshot. The Catalog.java Java class may indicate some errors, which are due to some required Maven dependencies not being added in the pom.xml file. We will add the required Maven dependencies in a later section.



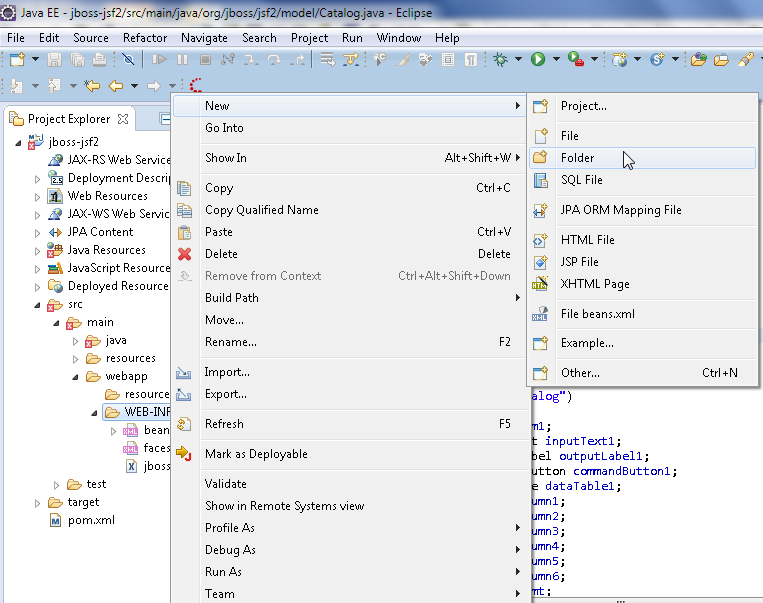
**[ 96 ]**

*Chapter 3*

## Creating a Facelets template

##### A Facelets template is a reusable component, which includes one or more JSF

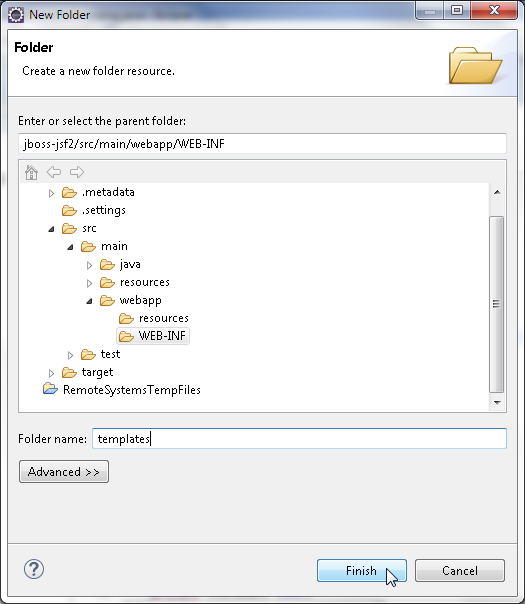
pages as different, sections of the template that may be used in multiple Facelets composition pages thus precluding the inclusion of each of the JSF pages that comprise the common sections of Facelets composition pages separately. We will use a Facelets template to include a header JSF page and a footer JSF page. The Facelets templates are to be created in the WEB-INF/templates directory for which you need to add a templates directory. Right-click on the WEB-INF folder in **Project Explorer** and select **New** | **Folder**, as shown in the following screenshot:



**[ 97 ]**

*Developing JSF 2.x Facelets*

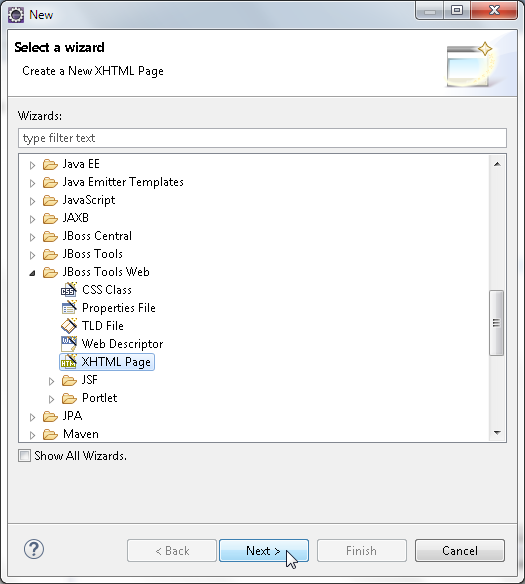
In the **New Folder** wizard, select the webapp | WEB-INF folder and specify **Folder name** as templates, as shown in the following screenshot:



**[ 98 ]**

*Chapter 3*

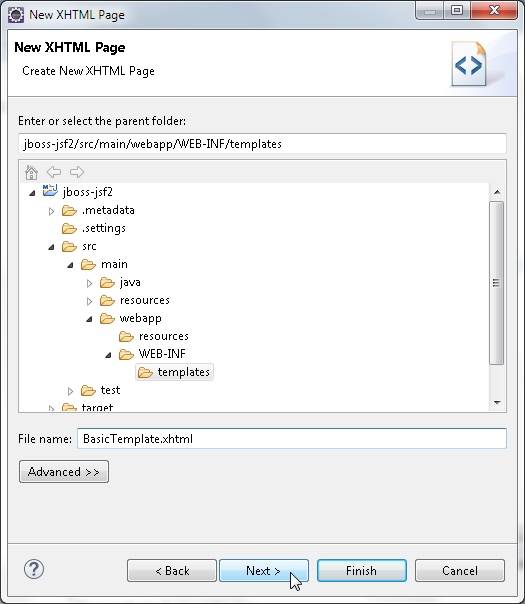
To create a Facelets template, select **File** | **New** | **Other**. In **New**, select **JBoss Tools Web** | **XHTML Page** and click on **Next**, as shown in the following screenshot:



**[ 99 ]**

*Developing JSF 2.x Facelets*

In **New XHTML Page** wizard, select the folder as webapp/WEB-INF/templates and specify **File name** as BasicTemplate.xhtml. Click on **Next**, as shown in the following screenshot:

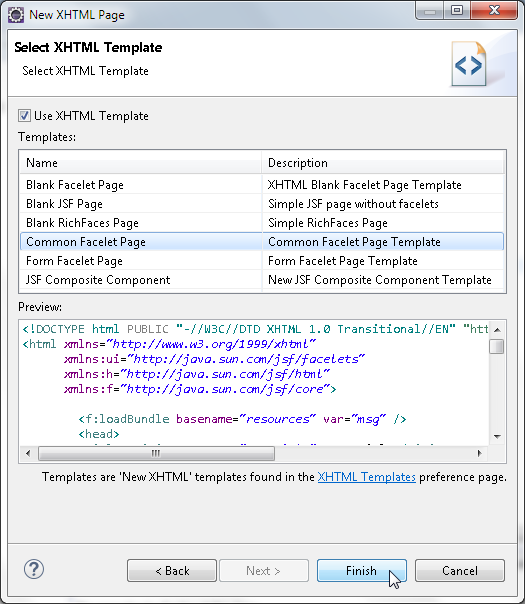


**[ 100 ]**

*Chapter 3*

In **Select XHTML Template**, select the **Common Facelet Page** template and click on

##### **Finish**, as shown in the following screenshot:



**[ 101 ]**

*Developing JSF 2.x Facelets*

##### The BasicTemplate.xhtml Facelet template gets created in the WEB-INF/templates folder. In the template, create <div/> elements for header, content, and footer sections of a Facelet composition page. The <ui:insert/> Facelets tag is used as a placeholder for a composition page section. As common header and footer sections are required in the input and output composition pages, you need to include a header JSF page in the header div and a footer JSF page in the footer <div> using the <ui:include/> tag.

Keep the <ui:insert/> element for the content div empty for the composition page to include the page section. For example, include a header.xhtml JSF page as follows:

<ui:insert name="header">

<ui:include src="/WEB-INF/templates/header.xhtml" />

</ui:insert>

We will create the header.xhtml and footer.xhtml JSF pages in the next section.

The BasicTemplate.xhtml template is listed as follows:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:ui="http://java.sun.](http://java.sun/) com/jsf/facelets">

<head>

<title><ui:insert name="title">JSF 2.0 Facelets</ui:insert></ title>

</head>

<body>

<div id="header">

<ui:insert name="header">

<ui:include src="/WEB-INF/templates/header.xhtml" />

</ui:insert>

</div>

<div id="content">

<ui:insert name="content">

</ui:insert>

</div>

<div id="footer">

<ui:insert name="footer">

<ui:include src="/WEB-INF/templates/footer.xhtml" />

</ui:insert>

</div>

</body>

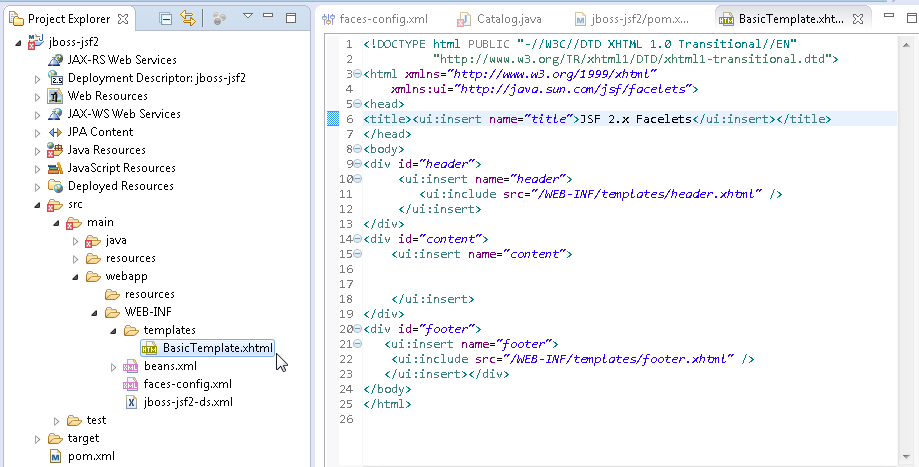
</html>

**[ 102 ]**

*Chapter 3*

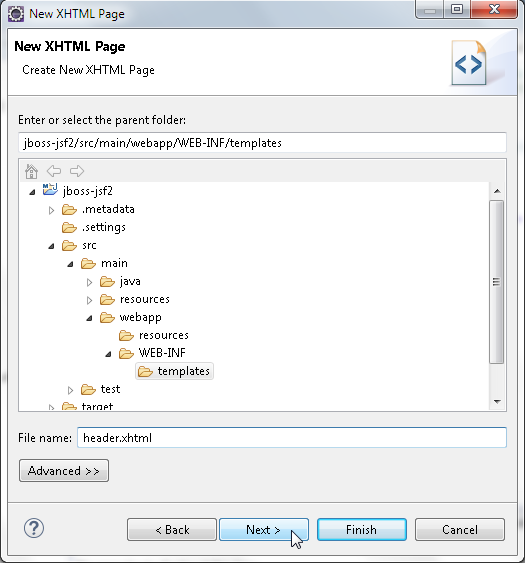
Copy the listing to the BasicTemplate.xhtml file in Eclipse IDE.

The BasicTemplate.xhtml file is shown in the following screenshot:



## Creating header and footer

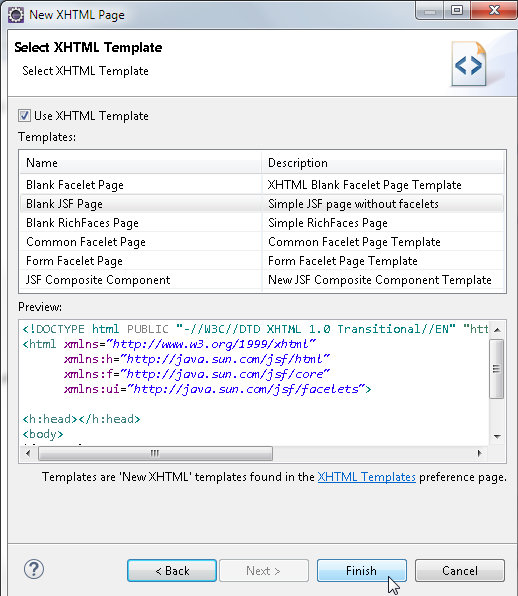
In this section, we will create header and footer JSF pages. Similar to creating the BasicTemplate.xhtml Facelet template, create header.xhtml and footer.xhtml in the WEB-INF/templates folder. For example, for the header JSF page, specify header.xhtml, as shown here:



**[ 103 ]**

*Developing JSF 2.x Facelets*

The difference in creating the header and footer pages that in **Select XHTML Template**, select the **Blank JSF Page** template shown as follows:



We will include graphic JPEG files for the header and footer sections. In the header. xhtml, include a JPEG file using the h:graphicImage tag enclosed in a h:panelGrid tag. First, copy the graphic JPEG files FaceletsHeader.jpg and FaceletsFooter. jpg to the //jboss-jsf2/src/main/webapp directory. The header.xhtml JSF page is listed as follows:

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:ui="http://java.sun.com/jsf/facelets"](http://java.sun.com/jsf/facelets) [xmlns:h="http://java.sun.com/jsf/html"](http://java.sun.com/jsf/html) [xmlns:f="http://java.sun.com/jsf/core">](http://java.sun.com/jsf/core)

<f:view>

<h:form>

<h:panelGrid columns="1">

<h:graphicImage value="FaceletsHeader.jpg" />

**[ 104 ]**

*Chapter 3*

</h:panelGrid>

</h:form>

</f:view>

</html>

Similarly, in the footer.xhtml JSF page, include a FaceletsFooter.jpg image file with the h:graphicImage tag enclosed in the h:panelGrid tag. The footer.xhtml page is listed as follows:

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:ui="http://java.sun.com/jsf/facelets"](http://java.sun.com/jsf/facelets) [xmlns:h="http://java.sun.com/jsf/html"](http://java.sun.com/jsf/html) [xmlns:f="http://java.sun.com/jsf/core">](http://java.sun.com/jsf/core)

<f:view>

<h:form>

<h:panelGrid columns="1">

<h:graphicImage value="FaceletsFooter.jpg" />

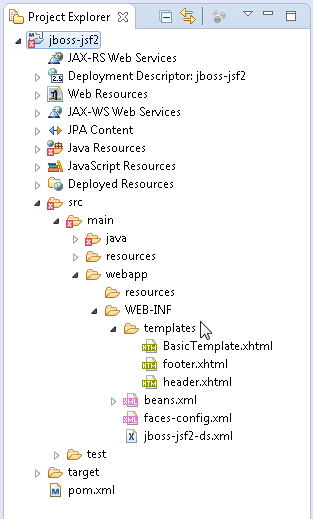
</h:panelGrid>

</h:form>

</f:view>

</html>

##### The directory structure of the jboss-jsf2 project is shown in the following screenshot:

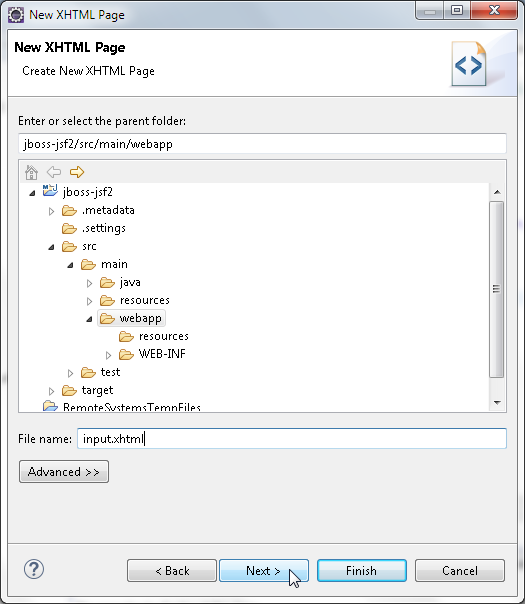


**[ 105 ]**

*Developing JSF 2.x Facelets*

## Creating input and output Facelets composition pages

In this section, we will create input and output Facelets composition pages, input.xhtml and output.xhtml. Create the Facelets pages similar to creating the BasicTemplate.xhtml, header.xhtml, and footer.xhtml to that select the folder to create input.xhtml and output.xhtml as webapp. For example, specify input. xhtml in **Create New XHTML Page**, as shown here:

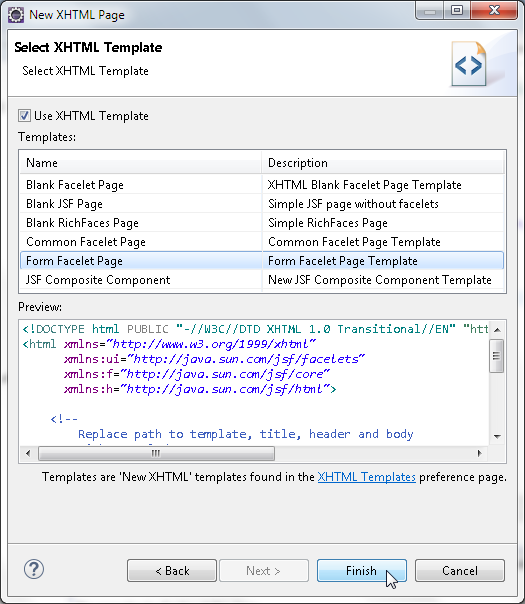


**[ 106 ]**

*Chapter 3*

And in **Select XHTML Template**, select **Form Facelet Page Template** and click on

##### **Finish**, as shown in the following screenshot:



**[ 107 ]**

*Developing JSF 2.x Facelets*

In the BasicTemplate.xhtml file, we defined the structure of a template that may

be reused in composition pages. The header and footer div tags are included in the header.xhtml and footer.xhtml files respectively. In the input.xhtml file, include the BasicTemplate.xhtml file using the ui:composition tag's template attribute. Specify the relative path to the template. We only need to define the content section of the input.xhtml composition page. The placeholder in the BasicTemplate. xhtml file is specified using <ui:insert name="content"/>. Specify the actual definition in the input.xhtml with <ui:define name="content"/>. Within the ui:define tag, add the JSF components for an input text with a corresponding output label, and a command button to invoke the action method in the managed bean catalog. The components have binding with corresponding managed bean properties using EL expression (<http://docs.oracle.com/javaee/6/tutorial/> doc/gjddd.html). Enclose the JSF components within h:panelGrid, which is enclosed within a h:form tag. The Facelet composition page is listed as follows:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:ui="http://java.sun.com/jsf/facelets"](http://java.sun.com/jsf/facelets) [xmlns:h="http://java.sun.com/jsf/html"](http://java.sun.com/jsf/html) [xmlns:f="http://java.sun.com/jsf/core">](http://java.sun.com/jsf/core)

<ui:composition template="/WEB-INF/templates/BasicTemplate.xhtml">

<ui:define name="content">

<h:form>

<h:panelGrid columns="2">

<h:outputLabel binding="#{catalog.outputLabel1}" value="SQL

Query:" />

<h:inputText binding="#{catalog.inputText1}" />

<h:commandButton value="Submit" binding="#{catalog. commandButton1}" action="#{catalog.commandButton1\_action}" />

</h:panelGrid>

</h:form>

</ui:define>

</ui:composition>

</html>

In the output.xhtml file, include BasicTemplate.xhtml with the template attribute of the ui:composition tag. Define the content section using the ui:define tag.

##### Within the ui:define tag, add a h:dataTable tag for a data table. Specify binding to the managed bean property dataTable1 using EL expression. Set the data table border with the border attribute of h:dataTable and set the number of rows to 5 using the rows attribute. Within the h:dataTable tag, add six h:column tags for the data table columns. Specify binding of the h:column tags to the managed bean properties.

**[ 108 ]**

*Chapter 3*

The output.xhtml page is listed as follows:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:ui="http://java.sun.com/jsf/facelets"](http://java.sun.com/jsf/facelets) [xmlns:h="http://java.sun.com/jsf/html"](http://java.sun.com/jsf/html) [xmlns:f="http://java.sun.com/jsf/core">](http://java.sun.com/jsf/core)

<ui:composition template="/WEB-INF/templates/BasicTemplate.xhtml">

<ui:define name="content">

<h:form>

<h:dataTable binding="#{catalog.dataTable1}" border="1" rows="5">

<h:column binding="#{catalog.column1}"></h:column>

<h:column binding="#{catalog.column2}"></h:column>

<h:column binding="#{catalog.column3}"></h:column>

<h:column binding="#{catalog.column4}"></h:column>

<h:column binding="#{catalog.column5}"></h:column>

<h:column binding="#{catalog.column6}"></h:column>

</h:dataTable>

</h:form>

</ui:define>

</ui:composition>

</html>

For a more detailed discussion on JSF 2 features, refer to *JavaServer Faces 2.0, Essential Guide for Developers*, *Cengage Learning*. Add an error.xhtml JSF page for displaying an error message. The error.xhtml page is not a Facelets composition page and just has a h:outputLabel tag with binding to the errorMsg managed bean property:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ["http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">](http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)

<html [xmlns="http://www.w3.org/1999/xhtml"](http://www.w3.org/1999/xhtml) [xmlns:h="http://java.sun.com/jsf/html"](http://java.sun.com/jsf/html) [xmlns:f="http://java.sun.com/jsf/core"](http://java.sun.com/jsf/core) [xmlns:ui="http://java.sun.com/jsf/facelets">](http://java.sun.com/jsf/facelets)

<head>

<title>Error Page</title>

</head>

<body>Error Page<h:outputLabel binding="#{catalog.errorMsg}" value="#{catalog.errorMsg}" />

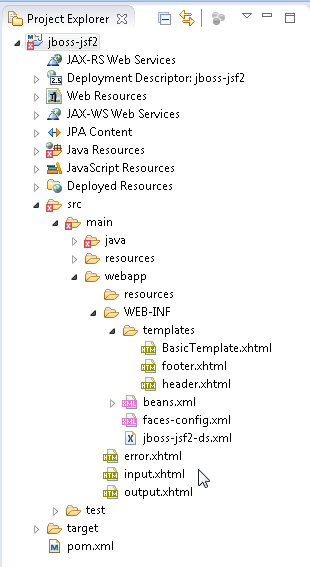
</body>

</html>

**[ 109 ]**

*Developing JSF 2.x Facelets*

The template BasicTemplate.xhtml, the header header.xhtml, the footer footer. xhtml and the input.xhtml and output.xhtml composition pages are shown in the **Project Explorer** tab:



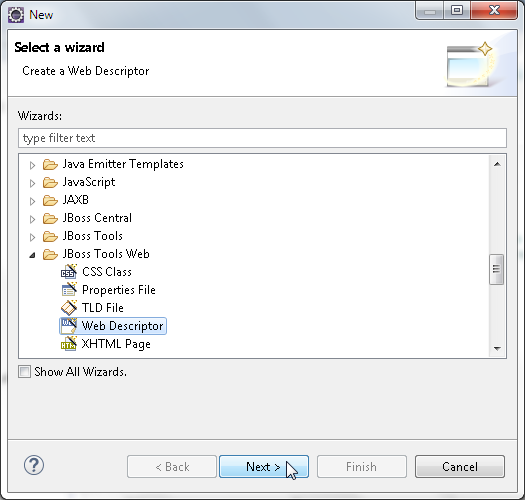
## Creating a web descriptor

##### A web descriptor (web.xml) is not a requirement in Java EE 7, but for a JSF application, we need to configure the Facelets servlet. To create a web descriptor, select **File** | **New** | **Other**.

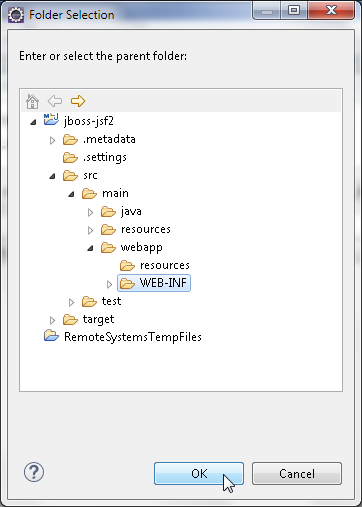
**[ 110 ]**

*Chapter 3*

In **New**, select **JBoss Tools Web** | **Web Descriptor** and click on **Next**, as shown in the following screenshot:



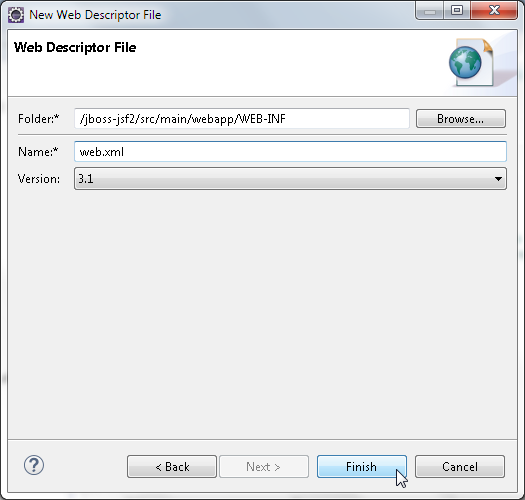
In the **Web Descriptor File** wizard for **Folder**, click on **Browse** to select a folder. In **Folder Selection**, select the **WEB-INF** folder and click on **OK**, as shown in the following screenshot:



**[ 111 ]**

*Developing JSF 2.x Facelets*

Select the **WEB-INF** folder and specify **Name** as web.xml, **Version** as 3.1 and click on **Finish**, as shown in the following screenshot:



##### A web.xml file gets created in the WEB-INF folder. In web.xml, specify the Faces servlet and its servlet mappings. The web.xml web descriptor is listed as follows:

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.1" [xmlns="http://xmlns.jcp.org/xml/ns/](http://xmlns.jcp.org/xml/ns/) javaee" [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

[xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee](http://xmlns.jcp.org/xml/ns/javaee) [http://xmlns.](http://xmlns/) jcp.org/xml/ns/javaee/web-app\_3\_1.xsd">

<display-name>JSF 2.x Facelets</display-name>

<servlet>

<servlet-name>Faces Servlet</servlet-name>

<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Faces Servlet</servlet-name>

<url-pattern>\*.jsf</url-pattern>

</servlet-mapping>

**[ 112 ]**

*Chapter 3*

<servlet-mapping>

<servlet-name>Faces Servlet</servlet-name>

<url-pattern>\*.faces</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>Faces Servlet</servlet-name>

<url-pattern>\*.xhtml</url-pattern>

</servlet-mapping>

<servlet-mapping>

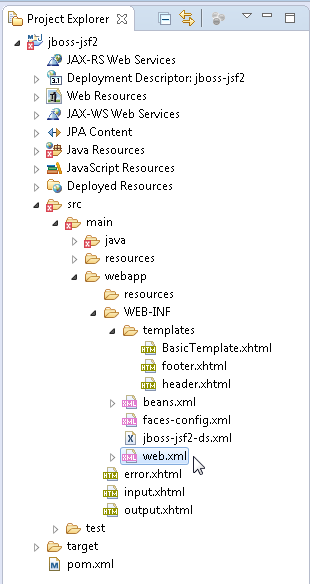
<servlet-name>Faces Servlet</servlet-name>

<url-pattern>/faces/\*</url-pattern>

</servlet-mapping>

</web-app>

The web.xml is shown in the following jboss-jsf2 web project:



**[ 113 ]**

*Developing JSF 2.x Facelets*

## Deploying the web project with Maven

##### The Java EE web project we created is based on Maven. It includes pom.xml to build, compile, and package the web application. The default project in WildFly 8.1 is a Java EE 7 version. As the Java EE project is a web project, the packaging gets specified

as war in pom.xml. The Group Id and Artifact Id attributes that we specified in creating a Java EE web project get configured in pom.xml as well. As we are using MySQL database, we need to add the dependency on the MySQL JDBC driver:

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.22</version>

</dependency>

##### The dependency on the Java EE 7 JSF 2.2 API, which is provided by WildFly 8.1, is

included in pom.xml by default.:

<dependency>

<groupId>org.jboss.spec.javax.faces</groupId>

<artifactId>jboss-jsf-api\_2.2\_spec</artifactId>

<scope>provided</scope>

</dependency>

##### As the managed bean and the Facelets composition pages use EL expressions, include a dependency on el-api, which is not provided by default by WildFly 8.1:

<dependency>

<groupId>javax.el</groupId>

<artifactId>javax.el-api</artifactId>

<version>3.0.0</version>

</dependency>

The **CDI** (**Context and Dependency Injection**) API, Common Annotations API,

JAX-RS API, **JPA** (**Java Persistence API**), **EJB** (**Enterprise JavaBeans**) API, Hibernate Validator API, annotation processor to generate the JPA metamodel classes, Hibernate validator annotation processor, and JUnit API are provided by WildFly

##### 8.1 by default and can be removed if not being used in the sample application in this chapter. In the build element, the Compiler plugin and the maven-war-plugin get configured by default. In the configuration for the maven-war-plugin plugin, specify the output directory for the WAR archive as the deployments directory of the WildFly 8.1 installation.

**[ 114 ]**

*Chapter 3*

The pom.xml file is listed as follows:

<?xml version="1.0" encoding="UTF-8"?>

<project [xmlns="http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) xmlns:xsi="http:// [www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) [xsi:schemaLocation="http://maven.](http://maven/) apache.org/POM/4.0.0 [http://maven.apache.org/maven-v4\_0\_0.xsd">](http://maven.apache.org/maven-v4_0_0.xsd)

<modelVersion>4.0.0</modelVersion>

<groupId>org.jboss.jsf2</groupId>

<artifactId>jboss-jsf2</artifactId>

<version>1.0.0</version>

<packaging>war</packaging>

<name>WildFly JSF 2.x</name>

<description>A starter Java EE 7 webapp project for use on JBoss WildFly / WildFly, generated from the jboss-javaee6-webapp archetype</ description>

[<url>http://wildfly.org</url>](http://wildfly.org/)

<licenses>

<license>

<name>Apache License, Version 2.0</name>

<distribution>repo</distribution>

[<url>http://www.apache.org/licenses/LICENSE-2.0.html</url>](http://www.apache.org/licenses/LICENSE-2.0.html)

</license>

</licenses>

<properties>

<!-- Explicitly declaring the source encoding eliminates the following message: -->

<!-- [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent! -->

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<!-- JBoss dependency versions -->

<version.wildfly.maven.plugin>1.0.2.Final</version.wildfly.maven. plugin>

<!-- Define the version of the JBoss BOMs we want to import to specify tested stacks. -->

<version.jboss.bom>8.1.0.Final</version.jboss.bom>

<version.arquillian.container>8.1.0.Final</version.arquillian. container>

<!-- other plugin versions -->

<version.compiler.plugin>3.1</version.compiler.plugin>

<version.surefire.plugin>2.16</version.surefire.plugin>

<version.war.plugin>2.1.1</version.war.plugin>

<!-- maven-compiler-plugin -->

<maven.compiler.target>1.7</maven.compiler.target>

<maven.compiler.source>1.7</maven.compiler.source>

**[ 115 ]**

*Developing JSF 2.x Facelets*

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-tools</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-hibernate</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.22</version>

</dependency>

<dependency>

<groupId>javax.el</groupId>

<artifactId>javax.el-api</artifactId>

<version>3.0.0</version>

</dependency>

<!-- Import the JSF API, we use provided scope as the API is included in JBoss WildFly -->

<dependency>

<groupId>org.jboss.spec.javax.faces</groupId>

<artifactId>jboss-jsf-api\_2.2\_spec</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

**[ 116 ]**

*Chapter 3*

<!-- Maven will append the version to the finalName (which is the name given to the generated war, and hence the context root) -->

<finalName>${project.artifactId}</finalName>

<plugins>

<!-- Compiler plugin enforces Java 1.6 compatibility and activates annotation processors -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>${version.compiler.plugin}</version>

<configuration>

<source>${maven.compiler.source}</source>

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>${version.war.plugin}</version>

<configuration>

<outputDirectory>C:\wildfly-8.1.0.Final\standalone\ deployments</outputDirectory>

<!-- Java EE 7 doesn't require web.xml, Maven needs to catch

up! -->

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

<!-- The WildFly plugin deploys your war to a local WildFly container -->

<!-- To use, run: mvn package wildfly:deploy -->

<plugin>

<groupId>org.wildfly.plugins</groupId>

<artifactId>wildfly-maven-plugin</artifactId>

<version>${version.wildfly.maven.plugin}</version>

</plugin>

</plugins>

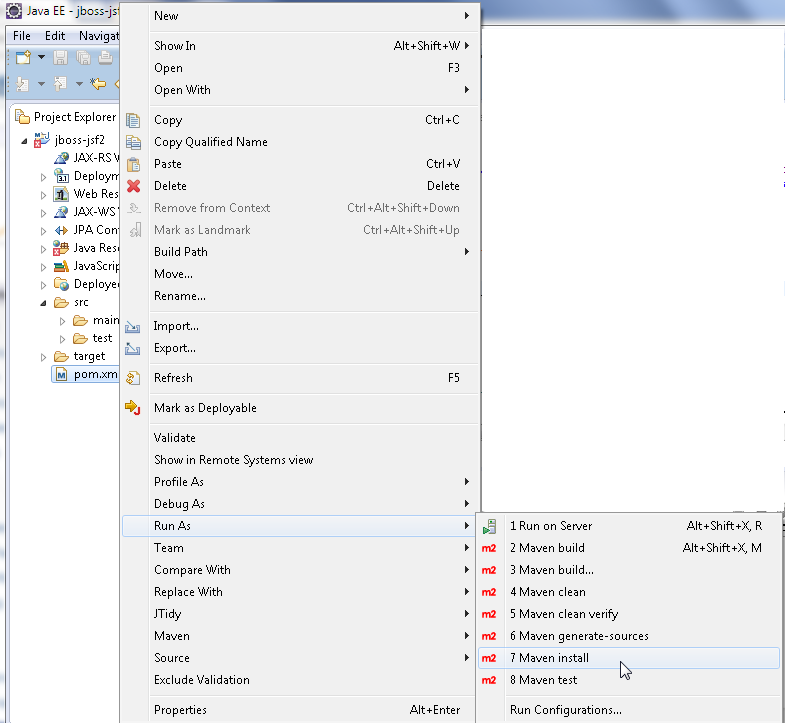
</build>

</project>

**[ 117 ]**

*Developing JSF 2.x Facelets*

Right-click on the pom.xml file and select **Run As** | **Maven install**, as shown in the following screenshot:

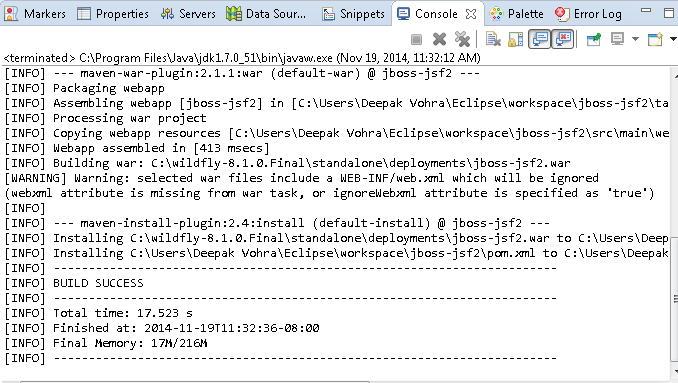


**[ 118 ]**

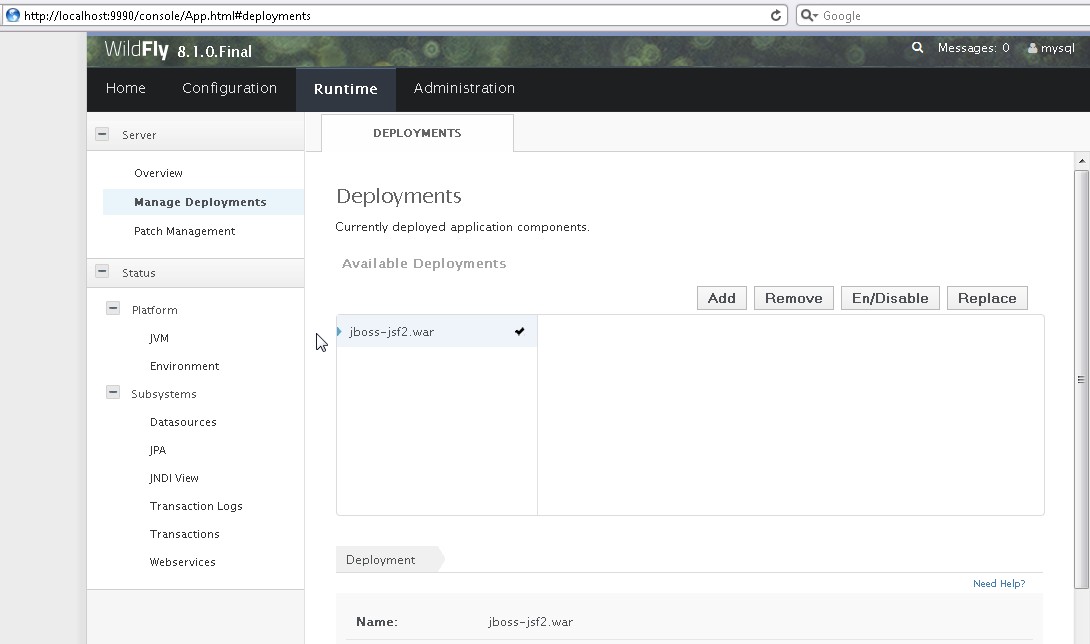
*Chapter 3*

The Maven pom.xml build completes with a BUILD SUCCESS message, as shown in the following screenshot. The jboss-jsf2 application gets compiled, packaged, and copied to the deployments folder, as all WAR archives copied to the deployments

##### directory get installed automatically and jboss-js2.war gets deployed to WildFly 8.1.



Navigate to the **Adminstration Console** in WildFly 8.1 using the URL http://localhost:8080/. The jboss-jsf2.war archive is listed as deployed, as shown in the following screenshot:

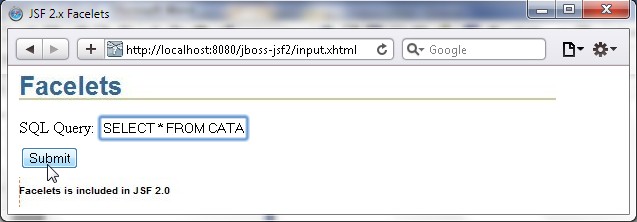


**[ 119 ]**

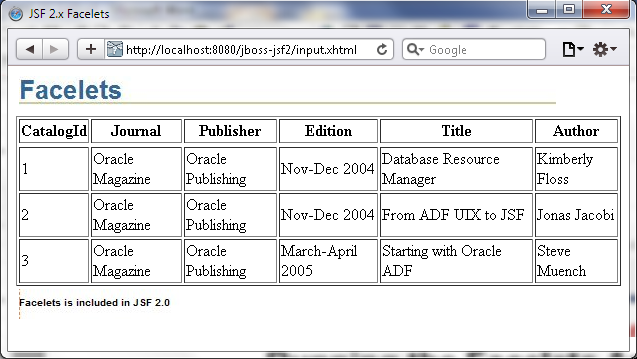
*Developing JSF 2.x Facelets*

## Running the Facelets application

In this section, we will run the JSF 2 application on WildFly 8. Invoke the input. xhtml file using the URL http://localhost:8080/jboss-jsf2/input.xhtml. The jboss-jsf2 is included as it is the context root for the jboss-jsf2.war application. The header and footer graphics JPEG files are included in the Facelets composition page using templating. Specify an SQL query, for example, SELECT \* FROM CATALOG and click on **Submit**, as shown in the following screenshot:



##### The input.xhtml page invokes the Facelets Servlet as .xhtml is specified in the servlet mapping. The action method of the managed bean generates a JSF data table and returns the output, which renders output.xhtml. The URL displayed in the browser stays the same because the request dispatcher sends a request forward, which does not start a new request. To display the output.xhtml file in the browser URL, a redirect will be required, which starts a new request. The same header and footer are included in the output.xhml file using templating, as shown in the following screenshot:



**[ 120 ]**

## Summary

*Chapter 3*

##### In this chapter, we created a JSF 2.x application to create a data table from an SQL query. We used Facelets based templating to include the same header and footer. The JSF 2.0 application is compiled and packaged using the Maven build tool and deployed to WildFly 8.1. The Facelets application is run on a browser to generate a data table.

In the next chapter, we will discuss using Ajax with WildFly 8.1.