



C

Working with NetBeans IDE 7.4

In the recent years, there has been a paradigm shift in the complexity and functionality of Java. The primitive approach to develop programs in Java has been to write your Java code in any text editor and then use the compiler in Java Development Kit (JDK) for compilation. This works well in case of simple programs, but would be a tiresome task for complex ones. A good development environment can have a tremendous impact on the creativity and productivity of the programmer. Therefore, this gave rise to the need of developing an environment that could handle the complexities and carry out the execution of all the modules of a Java application under one roof. Sun Microsystems identified the need of developing such an environment in 1999, and acquired the NetBeans Integrated Development Environment (IDE), which was particularly introduced for writing, compiling, and executing Java programs.

NetBeans is a free, open-source, cross-platform IDE with built-in support for Java. If a user uses IDE, it is mandatory for the user to work inside a project. An IDE project contains Java source files and associated information regarding Java ARchive (JAR) files and tools on the classpath, process of building and running the project, and other related information. You should note that the information regarding a project is stored inside a project folder. The project folder contains an Ant build script (build.xml) and the properties file that manages the settings that are required to build and run the project.

In this appendix, you explore the new features of NetBeans IDE 7.4 and the different User Interface (UI) components of NetBeans IDE 7.4, such as Start Page, Services window, Files window, Projects window, and Toolbars. You also learn how to create, build, and run the Web, enterprise, and JavaServer Faces (JSF) applications by using NetBeans IDE 7.4.

New Features of NetBeans IDE 7.4

Different versions of NetBeans IDE have come up since its first release in 1999, the latest version being NetBeans IDE 7.4. This version has been released in October 15, 2013.

Some noteworthy new features of NetBeans IDE 7.4 are as follows:

- **HTML5 Development**—NetBeans IDE 7.4 provides the following new features in HTML 5 development:
 - Allows you to create a Cordova application using template. Cordova is a new type of application type introduced in HTML 5, which allows to develop a mobile application to use standardised Web APIs on Android or iOS simulator or device
 - Provides the support for Android and iOS based browsers
 - Provides the editing support for stylesheet languages: SASS and LESS

- Provides a browser switcher icon in the main IDE toolbar
- Allows you to edit the files (e.g. CSS files) in your Web application inside Chrome Developer Tools, and save the edited changes in NetBeans IDE, which are actually saved in the real file on the disk.
- Provides network monitor, which helps in analyzing REST related communication, WebSocket communication and any network requests which are recently failed.
- ❑ **JavaScript**—NetBeans IDE 7.4 provides the following new features in JavaScript development:
 - Provides editing support for AngularJS, Knockout and ExtJS frameworks
 - Provides navigator and code folding features in JSON files
 - Provides enhancements in code completion facility
- ❑ **Java**—NetBeans IDE 7.4 provides the following new features in Java development:
 - Provides preview support for JDK 8 features, such as profiles and lambdas
 - Provides improvements in the editor, such as code completion, hints and refactoring
 - Allows native packaging
 - Provides updated versions of Ant 1.9.1 and Maven 3.0.5 bundling
- ❑ **JavaFX**—NetBeans IDE 7.4 provides the following new features in JavaFX development:
 - Provides a new dialog box, FXML File, for developing maven projects
 - Provides tighter alignment for JavaSE and JavaFX project types
 - Provides various project deployment options, such as defining custom manifest entries and building native packages
- ❑ **Java EE**: NetBeans IDE 7.4 provides the following new features in Java EE development:
 - Provides HTML5 features for Java EE projects, such as browser selection testing and visual CSS editing
 - Provides various wizards for JSF 2.2 and FacesComponents, such as Resource Library Contract wizard and @FacesComponent wizard
- ❑ **PHP**: NetBeans IDE 7.4 provides the following new features in PHP development:
 - Provides support of HTML5 features for PHP applications
 - Provides support for Nette Framework 2 and Zend Framework 2
 - Provides support for Atoum testing framework
 - Provides improvements in editor and rename type refactoring
 - Provides support for static code analysis
- ❑ **C/C++**—NetBeans IDE 7.4 provides the following new features in C or C++ development:
 - Provides support Run/Debug launchers
 - Provides improvements in find usages
 - Provides support for formatting style for C/C++ projects

In the next section, let's learn about the Start Page in NetBeans IDE 7.4.

The Welcome Page

When you launch NetBeans IDE 7.4, the first screen that appears is the Start Page. You can start a new project by selecting the File→New Project option from the menu bar. Several other menu options present in NetBeans IDE 7.4 are similar to the ones that were already available in the earlier versions of NetBeans. Figure C.1 shows the Start Page of NetBeans IDE 7.4:

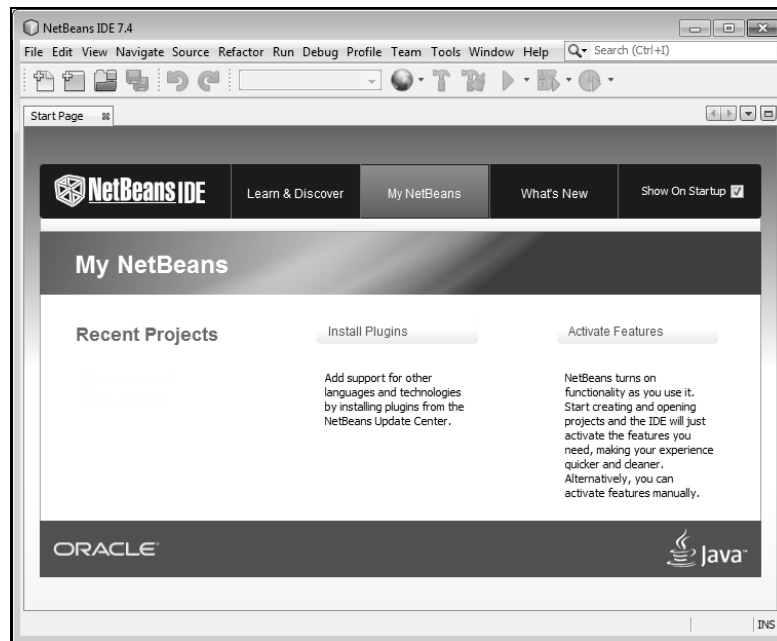


Figure C.1: Showing the NetBeans IDE 7.4 Start Page

The menu bar shown in Figure C.1 shows the various menus available in the NetBeans IDE.

Toolbars

Toolbars appear at the top of the NetBeans IDE, which allows you to implement functionality in an application or a project (Figure C.2). You can add and remove tools to and from the toolbars by clicking the View→Toolbars→Customize menu option in the IDE. The NetBeans IDE also provides the functionality to display a label (tooltip) while pointing the mouse to the menu. Many options in the toolbars get enabled or disabled, depending on the requirements of the project on which you are working. Figure C.2 shows the NetBeans IDE toolbars:

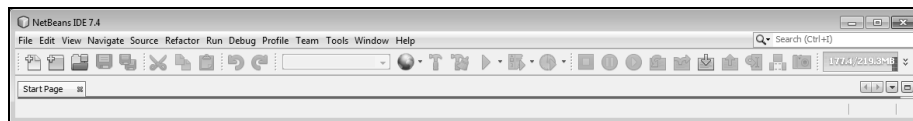


Figure C.2: Showing the NetBeans IDE 7.4 Toolbars

Although it is common to use the Menu system, the toolbar buttons provide quicker access. You can save the file on which you are currently working with the help of the diskette button (📁) on the Standard toolbar. To save all the files at a particular location in the computer, click the stacked diskettes button (📁).

The Projects Window

The Projects window appears on the upper left corner in the NetBeans IDE 7.4 (Figure C.3). It provides a tree view of all the projects on which you have worked recently. With the help of this window, you can browse through a particular project to access the different packages and libraries present in the project, such as Source Packages and Libraries. Figure C.3 shows the Projects window of NetBeans IDE 7.4:

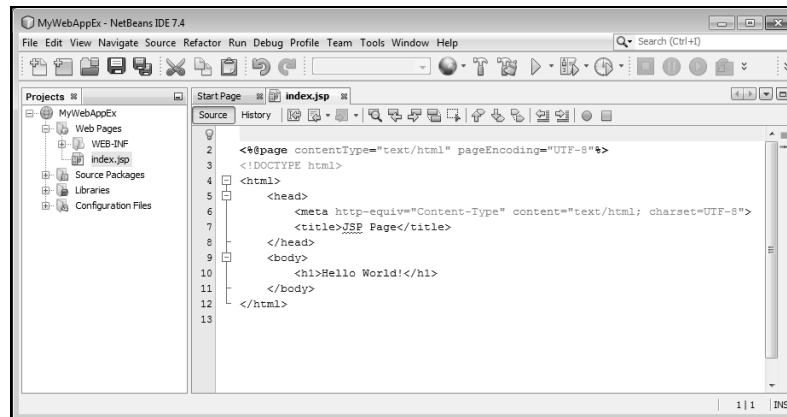


Figure C.3: Showing the Projects Window of NetBeans IDE 7.4

The Projects window is arranged in a parent-child tree manner, in which the parent node is the project and the child nodes are the categories of the project. For example, in Figure C.3, MyWebAppEx is the parent node containing various child nodes, such as Web Pages, Source Packages, and Libraries. Table C.1 describes the categories of child nodes in the parent node:

Table C.1: Describing the Categories of Various Child Nodes	
Node	Description
Web Pages	Contains various Web pages and the WEB-INF folder of an application. In the Web Pages node, you can add HyperText Markup Language (HTML), JSP, Cascading Style Sheets (CSS), and images for your application.
Source Packages	Permits you to access the Java source files of the project by expanding the nodes available beneath the Source Packages node.
Libraries	Specifies the libraries that are used in an application.
Configuration Files	Contains web.xml, struts-config.xml, and other project configuration files. The items available beneath the Configuration Files node are usually required to be accessed when configuration changes are made in the application. It also contains deployment descriptor for the Web application.

In the next section, let's learn about the Services window.

The Services Window

The Services window shares its space with the Projects window and provides various services, such as accessing different relational databases, configuring the servers, and managing the Web services of an application. The Services window can be opened by selecting the Services option under the Window menu. Figure C.4 displays the Services window of NetBeans IDE 7.4, which contains the list of different servers, databases, and services running under the IDE's control:

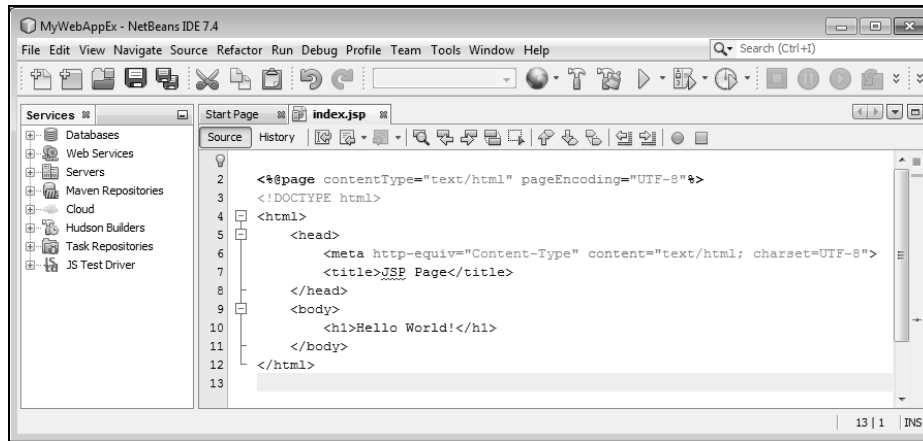


Figure C.4: Showing the Services Window of NetBeans IDE 7.4

The Files Window

The Files window of NetBeans IDE 7.4 displays a directory-based structure of the applications. This window also includes the files that are not displayed by the Projects window, such as the `build-impl.xml` file. The Files window lets you explore the various packages, files, and objects present in a project. Figure C.5 shows the Files window of NetBeans IDE 7.4:

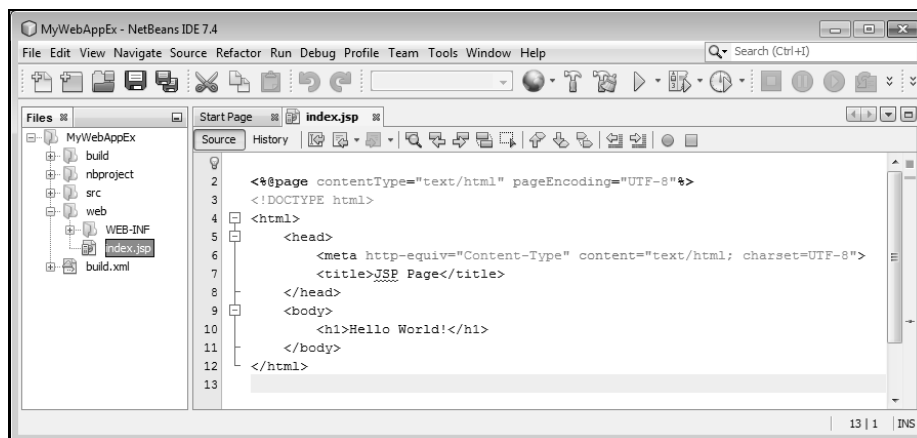


Figure C.5: Showing the Files Window of NetBeans IDE 7.4

You are now aware of the different types of window in NetBeans IDE 7.4. Next, let's learn how to create Web applications in the NetBeans IDE.

Creating Web Applications in the NetBeans IDE

Now, let's proceed to learn how to create a Java program in NetBeans. The NetBeans IDE provides different project templates to develop Java applications, Web applications, and enterprise applications. The two basic categories in which the available project templates can be divided are as follows:

- ❑ **Standard project**—Uses an IDE-generated Ant script to compile, run, and debug an application. An Ant script is an XML build file, which contains tasks that you want the Ant tool to perform. The following standard project templates are available in the IDE:
 - **Java**—Includes the Java Application, Java Class Library, Java Project with Existing Sources, and Java Free-Form Project options

- **Web**—Includes the Web Application, Web Application with Existing Sources, and Web Free-Form Application options
- **Enterprise**—Includes the Enterprise Application, Enterprise Application with Existing Sources, EJB Module, EJB Module with Existing Sources, Enterprise Application Client, and Enterprise Application Client with Existing Sources options
- **NetBeans Modules**—Includes the Module, Module Suite, and Library Wrapper Module options

Let's create a Web application in NetBeans IDE 7.4 by performing the following broad-level steps:

- ☐ Creating a Web Application in NetBeans IDE 7.4
- ☐ Modifying the default JavaServer Pages File
- ☐ Developing a Servlet
- ☐ Developing a Java Source File
- ☐ Developing a JavaServer Pages File

Creating a Web Application in NetBeans IDE 7.4

You can create a Web application in NetBeans IDE 7.4 by performing the following steps:

1. Select the **File**→**New Project** menu option to open the **New Project** wizard.

There are different categories of projects, such as Java, Java Web, Java EE, Java ME, Java FX, Maven, and many others, available in NetBeans IDE 7.4. To build a Java Web application, select **Java Web** from the **Categories** pane and **Web Application** from the **Projects** pane. Figure C.6 shows the **New Project** wizard:

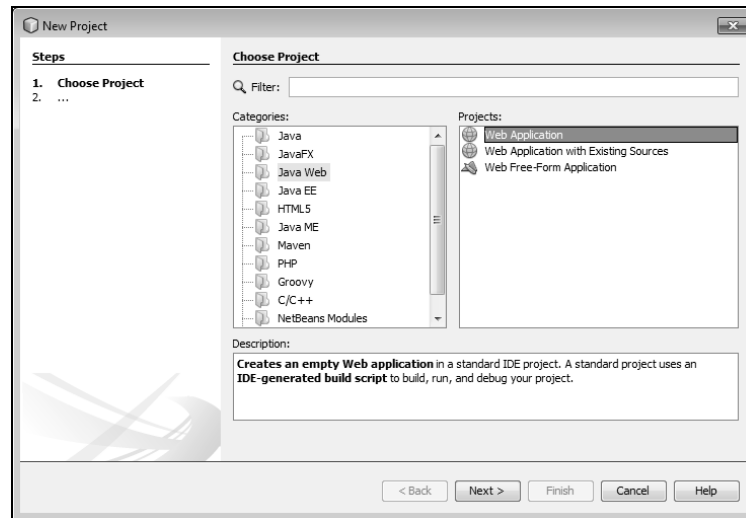


Figure C.6: Showing the New Project Wizard

2. Click the **Next** button.

The **Name and Location** page of appears in the **New Web Application** wizard, which provides the following options:

- **Project Name**— Allows you to enter the project name, which is used by NetBeans to display the project in its environment
- **Project Location**— Allows you to specify the project location by clicking the **Browse** button
- **Project Folder** - Contains the location of the project selected by the user or the default value of the project location

Figure C.7 shows the Name and Location page:

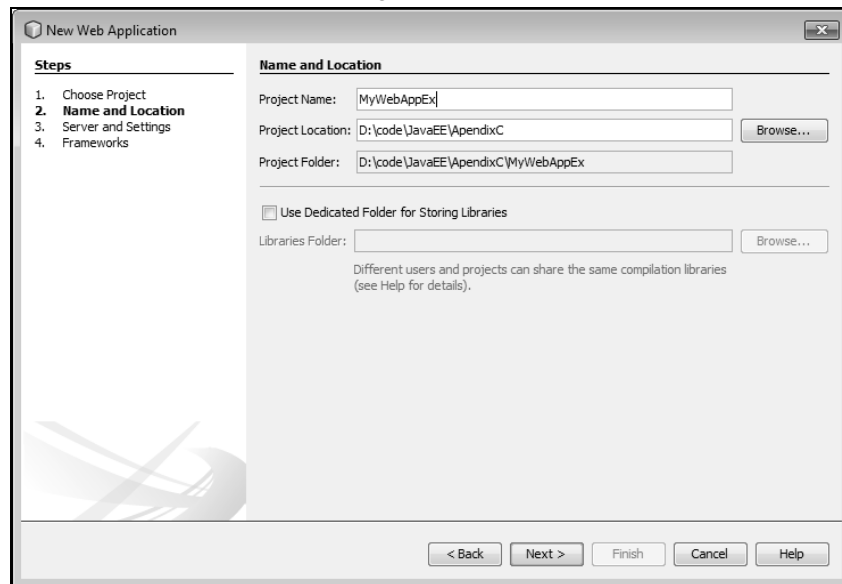


Figure C.7: Showing the Name and Location Page

3. Click the **Next** button (Figure C.7).

The **Server and Settings** page appears, which enables the user to select the desired application server for the Web application. The New Web Application wizard also enables the user to select the Java EE version and specify the context path of the Web application. Figure C.8 shows the **Server and Settings** page to select the desired application server, Java EE version, and provide context path:

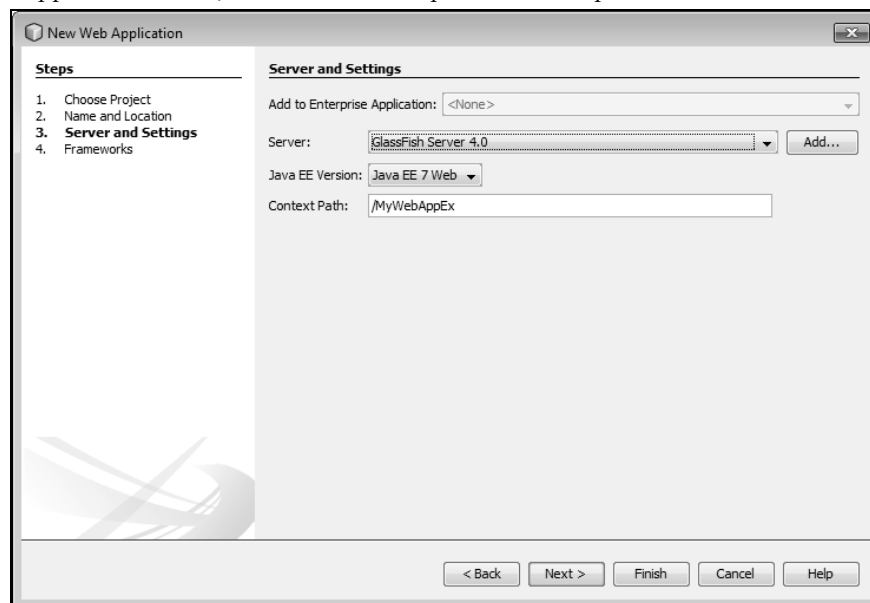


Figure C.8: Selecting the Appropriate Application Server

4. Click the **Next** button (Figure C.8).

The Frameworks page appears that allows you to select the desired framework for your Web application. If you do not want to implement any framework in your application, you can ignore the Frameworks page and click the Finish button. However, to select a framework, say JavaServer Faces, you need to select the checkbox beside the name of the framework. Figure C.9 shows the Frameworks page that displays frameworks to choose for the Web application:

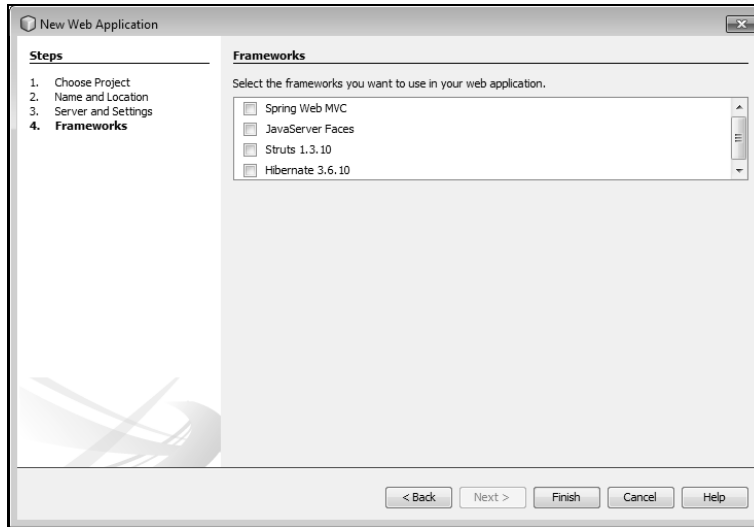


Figure C.9: Showing the Frameworks to Choose for the Web Application Project

5. Click the Finish button (Figure C.9) to proceed.

The `MyWebAppEx` project opens in the IDE and the default JSP page, i.e., `index.jsp`, is displayed in the Source Editor. Figure C.10 shows the `index.jsp` page:

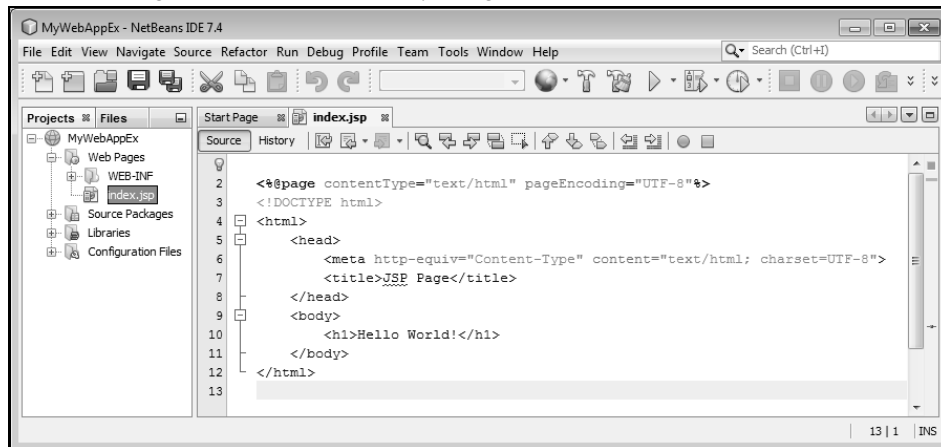


Figure C.10: Displaying the index.jsp Page in the Source Editor

Let's now learn how to edit the `index.jsp` page.

Editing the Default JavaServer Pages File

Perform the following steps to edit the JSP page:

1. Expand the HTML Forms tab in the Palette window (shortcut to open Palette window is CTRL+SHIFT+8), which is on the right of the Source Editor.

2. Drag the Form control from the Palette window and drop it just below the `<body>` tag in the Source Editor. The Insert Form dialog box opens, as shown in Figure C.11:

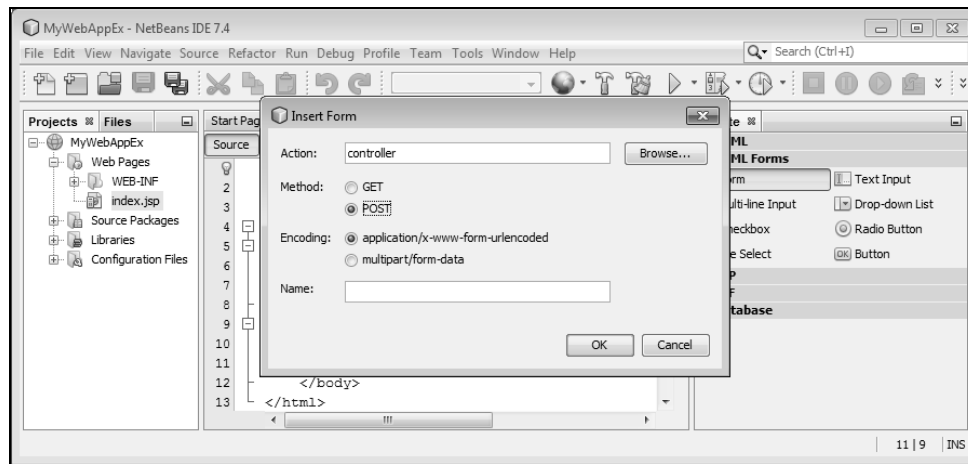


Figure C.11: Showing the Insert Form Dialog Box

3. Specify the values in the following fields of the Insert Form dialog box:
 - **Action**—Accepts the name of the component to be invoked. In our case, we have specified controller.
 - **Method**—Accepts the type of HTTP method, such as GET or POST. In our case, we have selected the POST radio button.
 - **Encoding**—Accepts the form encoding type that can be either application/x-www-form-urlencoded or multipart/form-data. In our case, we have selected the application/x-www-form-urlencoded radio button.
 - **Name**—Accepts the name of the form, which is optional. In our case, we have not provided any name for the form.
4. Click the OK button (Figure C.11). As a result, an HTML form tag `<form>....</form>` appears in the index.jsp file.
5. Drag the Drop-down List control from the Palette window to the Source Editor to a point just before the `</form>` tag. The Insert Drop-down List dialog box appears, wherein the following values are specified:
 - **Name**—Accepts the name of the component. In our case, we have specified genre as the name of the component.
 - **Options**—Accepts the number of options. In our case, we have specified 3 as the number of options to be provided in the drop-down list.
 - **Visible Options**—Accepts the number of visible options. In our case, we have specified 1 as the number of visible options in the drop-down list.
6. Click the OK button in the Insert Drop-down List dialog box (Figure C.12). An HTML `<select>` tag is added between the `<form>` tags.

Figure C.12 shows the Insert Drop-down List dialog box:

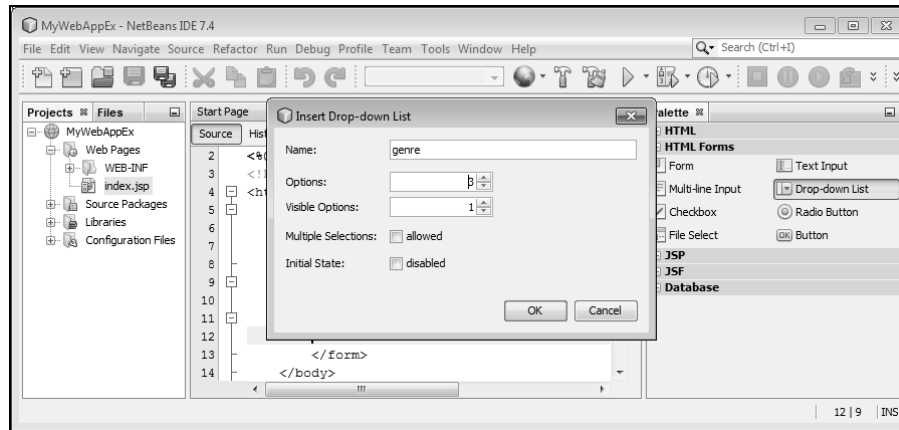


Figure C.12: Showing the Insert Drop-down List Dialog Box

7. Drag the Button control from the Palette window to a point just before the `</form>` tag. The Insert Button dialog box appears, which allows you to specify the following values:
 - **Label**—Allows you to specify the caption of the button. In our case, we have specified submit as the label.
 - **Type**—Allows you to specify the type of the button, such as submit, reset, or standard. In our case, we have specified submit as the type of the button.
 - **Name**—Allows you to specify the name of the button. In our case, we have specified button1 as the name of the button.

Figure C.13 shows the Insert Button dialog box:

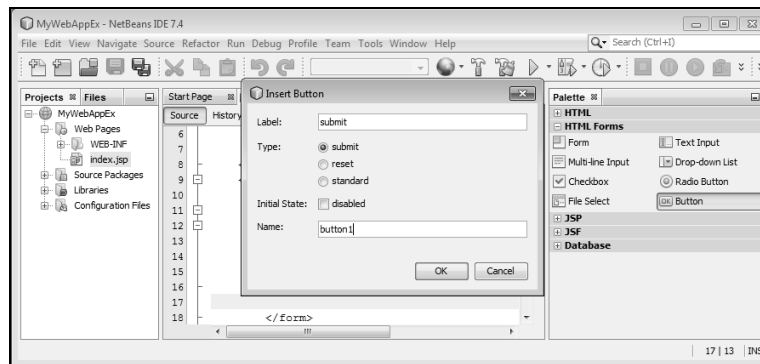


Figure C.13: Showing the Insert Button Dialog Box

8. Click the OK button (Figure C.13). The HTML button is appended in the `<form>` tag.
9. Right click the Source Editor and select the Format option to edit the code. Listing C.1 shows the code of the index page:

Listing C.1: Showing the Code of the index.jsp File

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
```

```

<body>
  <h1><b> Movie Selection Page</b></h1>
  <%
      String msg=(String)request.getAttribute("msg");
      out.print(msg+"<br>");
  %>
  <h2><b>Select your favourite Genre:</b></h2>
  genre:
  <form action="controller" method="POST">
  <select name="genre">
  <option>Comedy</option>
  <option>Sci-fi</option>
  <option>Action</option>
  </select>
  <input type="submit" value="submit" name="button1" />
  </form>
</body>
</html>

```

Creating the Servlet

To create a servlet, right click the MyWebAppEx application name in the **Projects** window, and choose the **New**→**Servlet** option. The **New Servlet** wizard opens. You need to provide the Class Name of the servlet as **controller** and specify **com.kogent.controller** as the Package name in the **New Servlet** wizard. Finally, click the **Finish** button. You will notice that a **controller.java** file node displays in the **Projects** window, and the new file opens in the **Source Editor**. Listing C.2 shows the code of the **controller.java** file:

Listing C.2: Showing the Code for the controller.java File

```

package com.kogent.controller;
import com.kogent.model.*;
import java.util.*;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(name="controller", urlPatterns={"/controller"})
public class controller extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
    response)
    throws ServletException, IOException
    {
        String myGenre=request.getParameter("genre");
        movieLists movies=new movieLists();
        List result=movies.getMovies(myGenre);
        RequestDispatcher view=null;
        if(result!=null)
        {
            request.setAttribute("result", result);
            view=request.getRequestDispatcher("result.jsp");
        }
        else
        {
            request.setAttribute("msg","Invalid Option: No Movies for such genre.");
            view=request.getRequestDispatcher("index.jsp");
        }
        view.forward(request, response);
    }
}

```

```
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the +
// sign on the left to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
/**
 * Returns a short description of the servlet.
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
}
}

```

Creating the Java Source File

Let's now create the Java source file, `movieLists.java`, by performing the following steps:

1. Right click the **Source Packages** node in the **Projects** window and select the **New→Java Class** option.
2. Enter `movieLists` in the **Class Name** field and specify `com.kogent.model` in the **Package** field.
3. Click the **Finish** button to complete the process of creating the Java source file.

The `movieLists.java` file will be opened in the **Source Editor**. Listing C.3 shows the code of the `movieLists` Java source file:

Listing C.3: Showing the Code of the `movieLists.java` File

```
package com.kogent.model;
import java.util.*;
public class movieLists {
    public List getMovies(String genre)
    {
        List movies=new ArrayList();
        if (genre.equals("Comedy"))
        {
            movies.add("I love you to death");
            movies.add("scary movie 1");
            movies.add("scary movie 2");
        }
        else if(genre.equals("Sci-fi"))
        {

```

```

        movies.add("matrix");
        movies.add("matric reloaded");
    }
    else
    {
        return null;
    }
    return (movies);
}
}

```

The Servlet will then forward the result to another JSP page called result.jsp. Let's learn to create result.jsp file in the next subsection.

Creating the JavaServer Pages File

In the **Projects** window, right click the project node (in our case, **MyWebAppEx**) and select the **New**→**JSP** option. The **New JSP wizard** opens. In the **File Name** field, enter **result** and click the **Finish** button to complete the process. Notice that a **result.jsp** file node displays in the **Projects** window. Listing C.4 shows the code of the result JSP page:

Listing C.4: Showing the Code of the result.jsp File

```

<%@page contentType="text/html" import="java.util.*"%>
<%@page pageEncoding="UTF-8"%>
<@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>check out these cool movies!!!</title>
  </head>
  <body>
    <h1>movies under selected Genres are:</h1>
    <c:set var="reqattr" scope="request" value="${result}" />
    <table>
      <c:forEach var="movie" items="${reqattr}">
        <tr>
          <td>
            ${movie}
          </td>
        </tr>
      </c:forEach>
    </table>
  </body>
</html>

```


Once all the required files are being coded, it's now time to build and run the application.

Building and Running the Web Application

To build the **MyWebAppEx** project, select the **Run**→**Build Project (MyWebAppEx)** option from the menu bar in the NetBeans IDE 7.4. After the project is built successfully, you can run the **MyWebAppEx** application by selecting the **Run Project (MyWebAppEx)** option from the **Run** menu.

NOTE

The following are the three ways of running a Web Application in NetBeans IDE 7.4:

- By selecting the **Run Project (MyWebAppEx)** option from the **Run** menu
- By clicking the **Run Project (MyWebAppEx)** button ()
- By pressing the **F6** key

When you run a Web application, the IDE builds the project first and then deploys it to the server that you had specified at the time of project creation.

Figure C.14 shows the execution of the **MyWebAppEx** application that displays the Movie Selection Page:



Figure C.14: Showing the Movie Selection Page

4. Select a genre, such as Comedy in our case (Figure C.14), and then click the submit button to check the movies under the selected genres.

Figure C.15 shows a list of movies under the selected genre (Comedy):

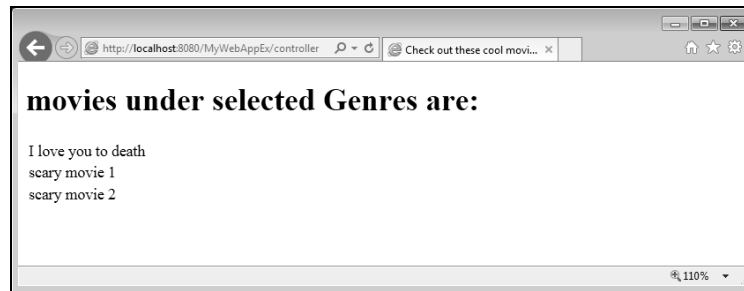


Figure C.15: Showing a Movies List under the Comedy Genre

Similarly, if you select another genre category, you will get another type of movie list specified for that particular genre. Moreover, the Invalid Option: No Movies for such genre message will be displayed if no movie list is available for a particular genre, such as Action. Figure C.16 shows the resultant Web page after selecting the Action category:



Figure C.16: Showing Invalid Option Message under the Action Genre

This was all about creating a sample Web application using NetBeans IDE 7.4. Let's now learn how to create the enterprise applications in NetBeans IDE 7.4.

Creating Enterprise Applications in NetBeans IDE

The enterprise application project is a collection of Web application and EJB modules that are configured to work together when deployed to the Java EE Application Server.

The enterprise application contains the information regarding integrated working of the modules and how the modules work with the application server. It contains deployment descriptors and other configuration files, but has no source files of its own. In compilation, the JAR files and WAR files for each individual module of the enterprise application are built and assembled into one Enterprise Archive (EAR) file, which is deployed to the application server. An enterprise application project can be created manually or from existing sources.

Creating a Enterprise Project in NetBeans IDE 7.4

You can create a new enterprise project by using NetBeans IDE 7.4. The following steps are performed to create the enterprise project in NetBeans IDE 7.4:

1. Select the **File**→**New Project** option from the menu bar to open the **New Project** wizard.
You will see a similar window that you saw when you create the **MyWebAppEx** Web application.
2. Select **Java EE** from the given category list and **Enterprise Application** from the given project list. To proceed, click the **Next** button.

Figure C.17 shows the **New Project** wizard containing the options for the category list and project list:

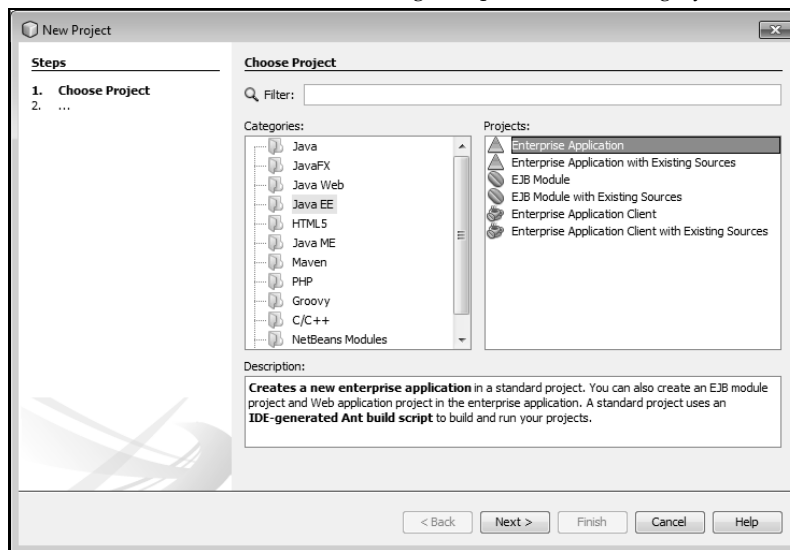


Figure C.17: Showing the New Project Wizard to Choose an Enterprise Application Project

The **New Enterprise Application** wizard appears.

3. Enter the details for the following fields in the **New Enterprise Application** wizard:
 - **Project Name**—Allows the user to enter the project name for the Enterprise Application project, and NetBeans will use this name to display the project in its environment. In our case, we have entered **EnterAppEx** as the project name.
 - **Project Location**—Allows the user to enter the project location by clicking the **Browse** button. It may also accept the default value if the user do not specify it. In our case, we have continued with the default value.
 - **Project Folder**—Contains the location of the application. In our case, we have continued with the **EnterAppEx** project folder located at the default location.

Figure C.18 shows the **New Enterprise Application** wizard:

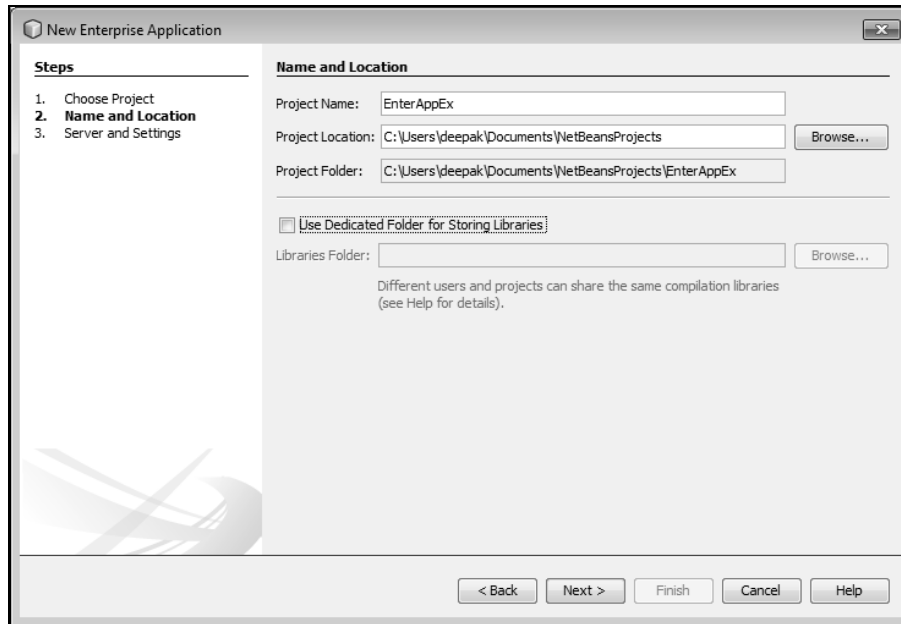


Figure C.18: Showing the New Enterprise Application Wizard

4. Click the **Next** button to proceed. The **Server and Settings** page appears that enables the user to select the desired application server and Java EE version for the enterprise application.

Figure C.19 shows the **Server and Settings** page to select the desired application server:

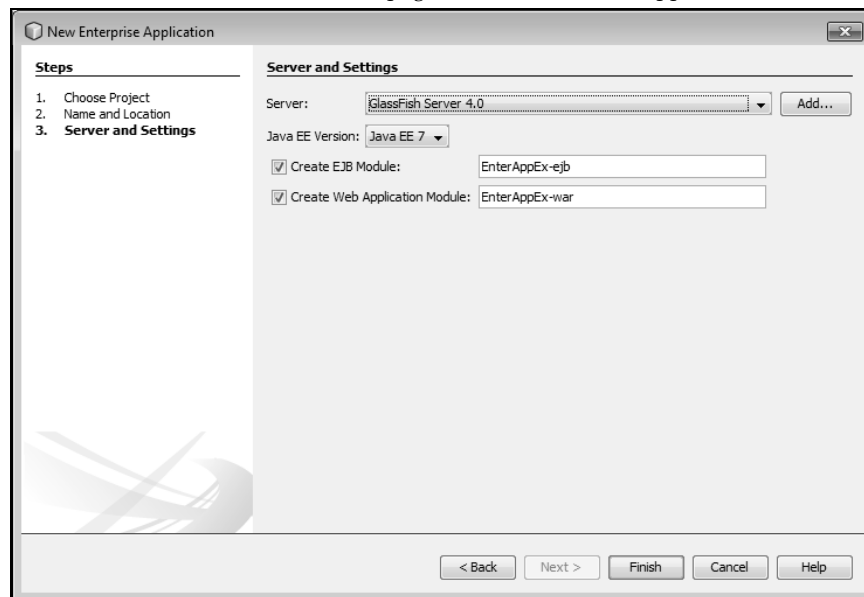


Figure C.19: Showing the Window to Choose Desired Application Server

5. Click the **Finish** button (Figure C.19) to create the enterprise application. The newly created enterprise application appears that displays the Web and EJB modules.

Figure C.20 displays the Web and EJB modules of the `EnterAppEx` application:

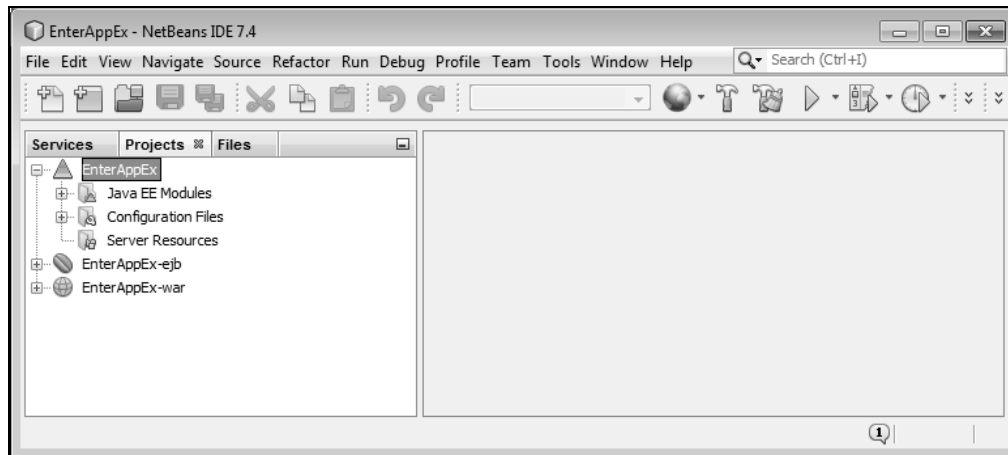


Figure C.20: Showing the Web and EJB Module of the EnterAppEx Application

Now you can provide code for the Web and EJB modules of the `EnterAppEx` enterprise application.

Creating JSF Applications in NetBeans IDE

The new Java framework, particularly used for creating Web applications, is JSF. JSF framework provides easier application development through component-centric approach for Java Web UI creation. Simplified and robust JSF API provides application developers unparalleled power and programming flexibility. To provide greater maintainability to the application, JSF framework's architecture also accommodates the Model-View-Controller (MVC) design pattern. You need to perform the following steps to create the JSF application in NetBeans IDE 7.4:

1. Create the Web application in NetBeans IDE 7.4
2. Develop the view pages
3. Develop the properties file
4. Develop a Java class
5. Develop the faces-config.xml file

Let's discuss about these steps, which are needed to develop a JSF application, in detail.

Creating a Web Application in NetBeans IDE 7.4

You can create a new Web application by using NetBeans IDE 7.4. The following steps are performed to create the Web application in NetBeans IDE 7.4:

1. You can create a new application by selecting the `File`→`New Project` menu option. The New Project wizard appears that is similar to the wizard displayed during the creation of the `MyWebAppEx` Web project.
2. For the JSF project, select the `Java Web` option from the given category list and the `Web Application` from the given project list.

Figure C.21 shows the New Project wizard:

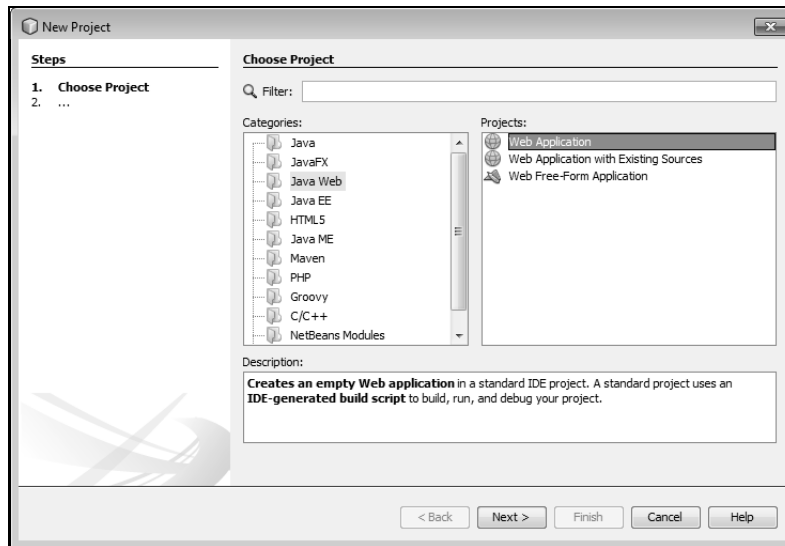


Figure C.21: Showing the New Project Wizard

3. Click the Next button (Figure C.21).

The New Web Application wizard appears.

Figure C.22 shows the New Web Application wizard:

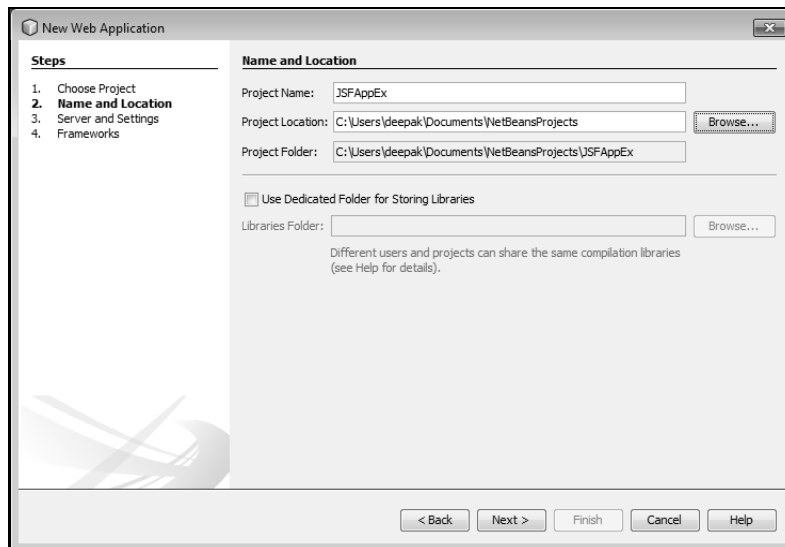


Figure C.22: Showing the New Web Application Wizard

The New Web Application wizard provides various options, which allow the user to enter the project name and project location. Provide the value `JSFAppEx` to the Project Name option. Click the Next button to proceed.

Now, the Server and Settings page appears that enables the user to select the desired application server, Java EE version, and specify the context path for the Web application.

Figure C.23 shows the page to select the desired application server:

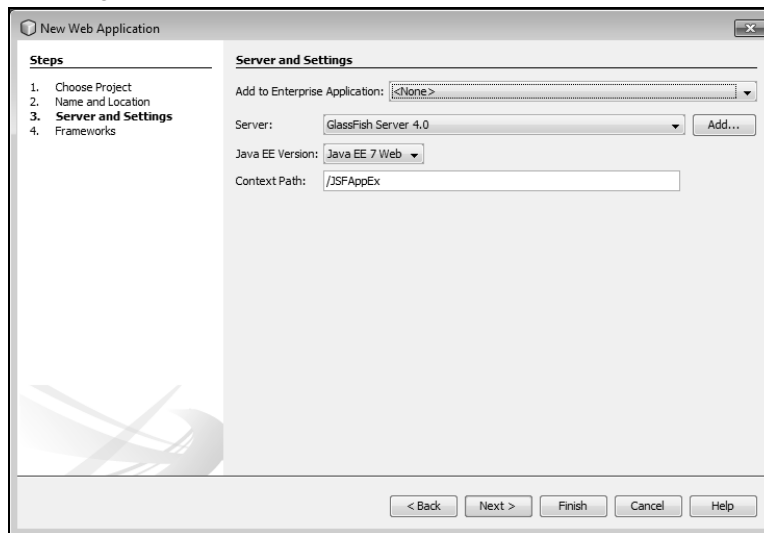


Figure C.23: Showing the Window to Select the Desired Application Server

4. Click the **Next** button (Figure C.23).

The **Frameworks** page appears that enables the user to select the appropriate framework. Check the **JavaServer Faces** check box to include the JSF framework in the application.

Figure C.24 shows the **Frameworks** page displaying the various frameworks that can be selected:

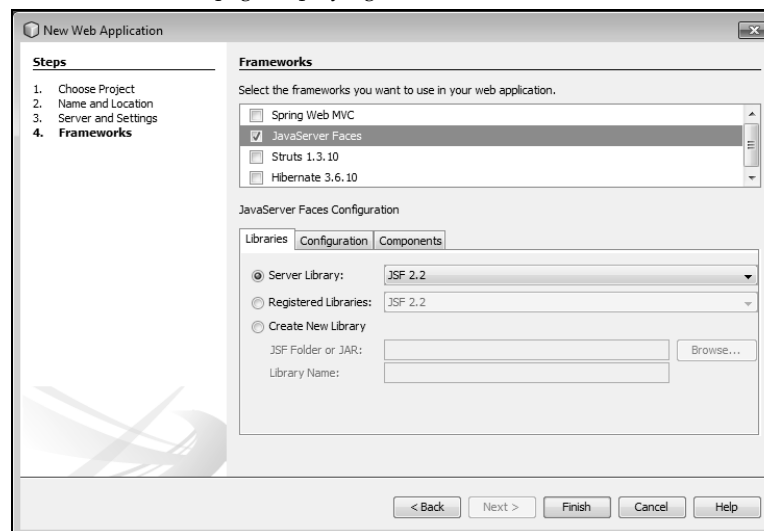


Figure C.24: Showing the Window to Select the Desired Framework

5. Click the **Finish** button (Figure C.24) to create the JSFAppEx application.

Creating the View Pages

Now it's the time to create view pages for the application. First, the welcome page (set the DemoMain.xhtml page as the welcome page in the web.config file) is created, which is DemoMain.xhtml file. To create an XHTML file, the following steps are performed:

1. Right click the project node, and select the New→Other→Web option. Now, select the XHTML file from the category list.

Figure C.25 shows the New File wizard that provides the XHTML option to create the XHTML file:

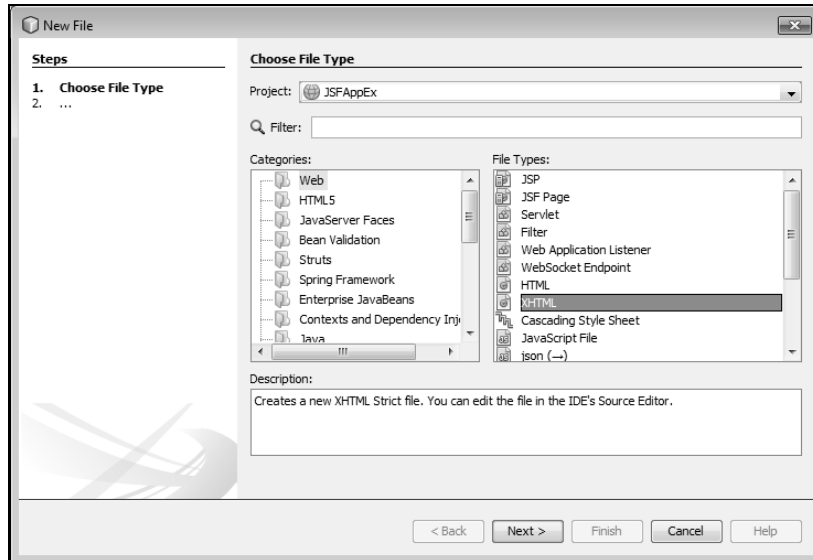


Figure C.25: Showing the Creation of XHTML File

2. Click the Next button (Figure C.25).

The Name and Location page appears that enables the user to enter the name of the XHTML file, which is DemoMain in the application.

Figure C.26 shows the Name and Location page to enter the name of the XHTML file:

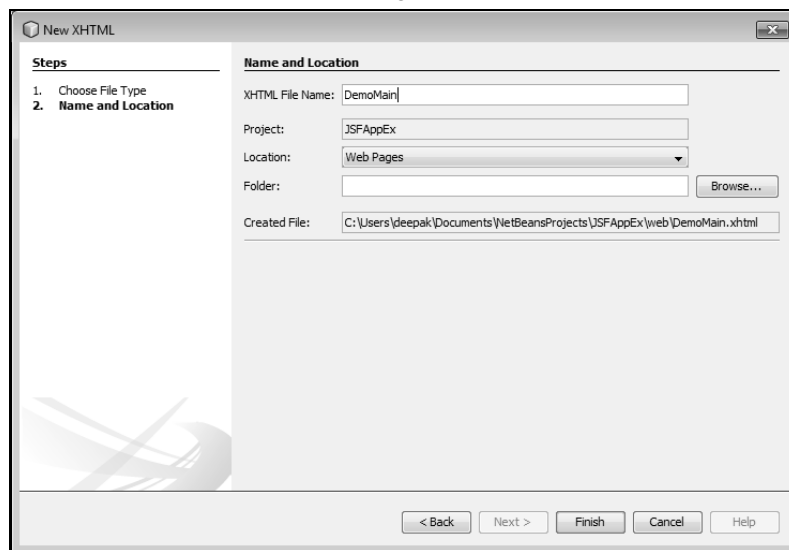


Figure C.26: Showing the Window to Enter the XHTML File Name

3. Click the Finish button (Figure C.26).

The `DemoMain.xhtml` file appears in the Projects window. Edit the code of the `DemoMain.xhtml` file according to the code provided in Listing C.5:

Listing C.5: Showing the Code of the `DemoMain.xhtml` File

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <body>
    <ui:composition template="/myTemplate.xhtml">
      <ui:define name="body">
        <h:form>
          <h1><h:outputText value="#{msgs.AppTitle}"/> </h1><br/>

          <h:outputText value="#{msgs>YourName}"/>    &#160;
            <h:inputText value="#{DemoBean.name}" /> <br/>
            <h:commandButton value="Press" action="DemoHello"></h:commandButton>
          </h:form>
        </ui:define>
      </ui:composition>
    </body>
  </html>
```

When the Web pages are developed in a Web application, there is some repeating content in multiple Web pages. Writing the repeating content again and again in the Web pages reduces the programmer productivity, thereby increasing the project time and cost. The solution to this problem is to use templates, which contain the repeating code and may be reused across multiple Web pages. Listing C.5 uses the `<ui:composition>` tag to invoke a template. The template attribute of the `<ui:composition>` tag specifies the template to invoke, which is `myTemplate.xhtml` in this application. The `<ui:define>` tag is utilized to pass the parameter to the template and is a sub element of the `<ui:composition>` tag. The following code snippet shows the line in Listing C.5, which is to be noticed:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
```

In the preceding code snippet, the namespace inclusion permits the page to use the JSF's facelet, html components. Listing C.5 uses the `<ui:define>` tag to invoke the section of template, which is defined with the `<ui:insert>` tag in the template page. The name attribute of the `<ui:define>` tag must match with the name attribute of the `<ui:insert>` tag in the template page. Listing C.5 uses the `<h:outputText>` tag to output a value. The value attribute of the tag signifies the value to print. The value `msgs` is the name of the property file, which is `messages.properties`, provided in the `faces-config.xml` JSF configuration file. The property file is discussed under the *Creating the Properties File* heading. You should note that `Hello` and `AppTitle` are the properties of the `messages.properties` file. `DemoBean` is the name of Java class in this application, which is discussed under the *Creating the Java Class* heading.

Listing C.6 provides the code of the `myTemplate XHTML` page:

Listing C.6: Showing the Code of the `myTemplate.xhtml` File

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <head>
  </head>
  <body>
    <h1>
      <ui:insert name="title"/>
    </h1>
    <p>
      <ui:insert name="body"/>
    </p>
  </body>
</html>
```

```

    </p>
  </body>
</html>

```

When the user submits the name in the DemoMain.xhtml file, the DemoHello.xhtml page is displayed to the user. The navigation in JSF is fully handled by the Web application, according to the rule specified in the faces-config.xml file. The DemoHello.xhtml page displays the message Hi name of the user Welcome to the JSF Application. Listing C.7 shows the code of the DemoHello XHTML page:

Listing C.7: Showing the Code of the DemoHello.xhtml File

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <body>
    <ui:composition template="/myTemplate.xhtml">
      <ui:define name="body">
        <h:outputText value="#{msgs.Hello}"/>
        &#160; <h:outputText value="#{DemoBean.name}"/>
        &#160; <h:outputText value="#{msgs.AppTitle}"/>
      </ui:define>
    </ui:composition>
  </body>
</html>

```

Creating the Properties File

The properties class is used by the view component to display the text. One of the uses of the property file is to avoid repetition of the same text in various pages within a Web application. The text to be displayed multiple times is written inside the properties file in the form of property with a key and a value. The value of a property in the properties file is the text that is displayed in the Web page. The key of the property is used in the Web pages to display the value of the property. To create a property file, the following steps are performed:

1. Right click the Source Packages node in the Projects window and select the New→Other option. The New File wizard appears (Figure C.27).
2. Select the Other→Properties File option in the New File wizard (Figure C.27).
3. Click the Next button, as shown in Figure C.27:

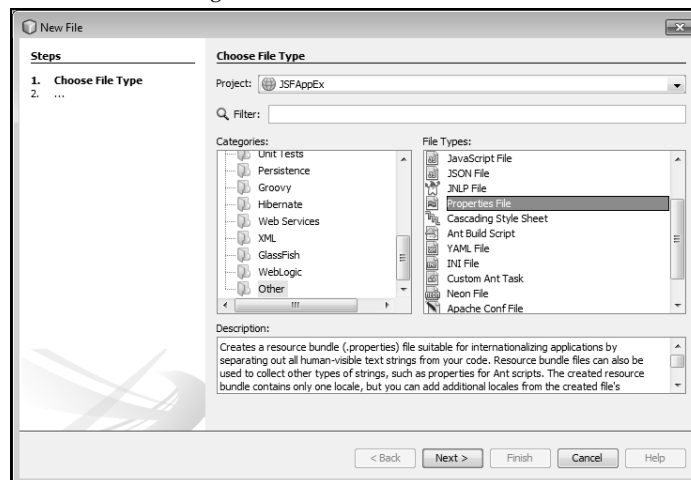


Figure C.27: Showing the New File Wizard

The `New Properties File` wizard appears. Enter the name of the properties file and folder as `messages` and `src\java\com\kogent` respectively, as shown in Figure C.28:

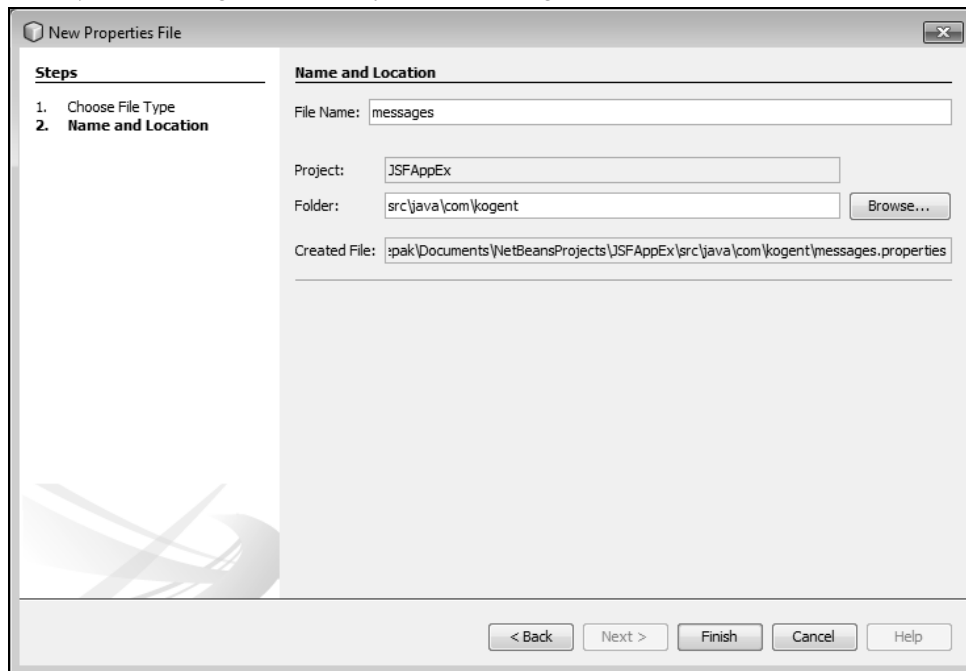


Figure C.28: Showing the Window to Enter Name and Location for a Properties File

4. Click the `Finish` button (Figure C.28).

The `messages.properties` file is created. To add the property to the property file, right-click the properties file node and select `Add→Property`. The `New Property` dialog box appears, which enables the user to enter the name and value of the property.

Figure C.29 shows the `New Property` dialog box to create the `Hello` property:

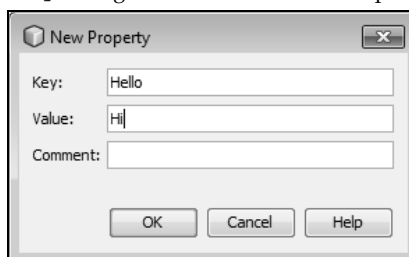


Figure C.29: Showing the New Property Dialog Box

Similarly, two other properties of the `messages.properties` file can be created, which are `YourName` and `AppTitle`.

The code of the `messages.properties` file is shown in Listing C.8:

Listing C.8: Showing the Code of the `messages.properties` File

```
Hello=Hi
AppTitle=welcome to the JSF Application
YourName=what is your name?
```

Creating the Java Class

You can create a Java class by performing the following steps:

Right click the project node and select the **New**→**Java Class** option.

1. The **New Java Class** wizard opens, which enables the user to enter the name of class and package. Provide the value **DemoBean** in the **Class Name** text box and **com.kogent** in the **Package** text box.

Figure C.30 shows the **New Java Class** wizard:

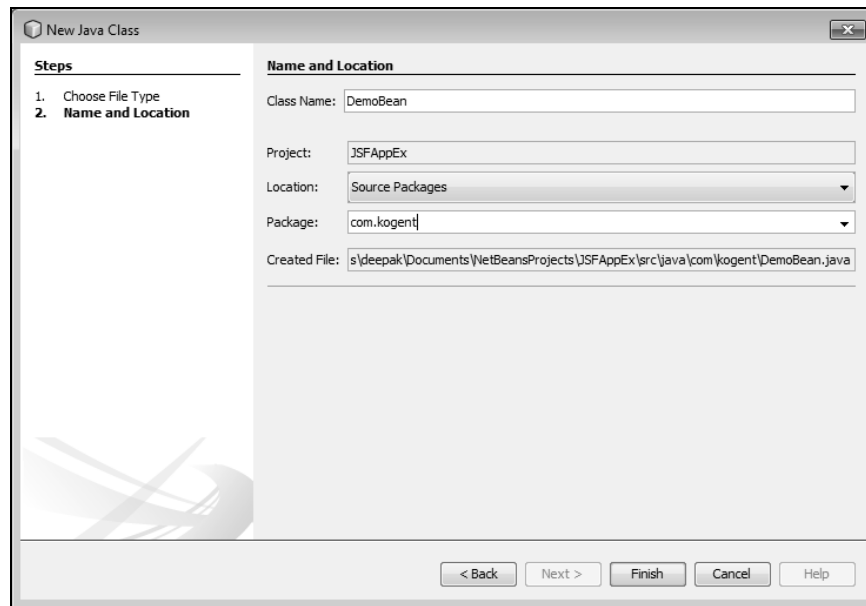


Figure C.30: Showing the New Java Class Wizard

2. Click the **Finish** button to create the **DemoBean** class (Figure C.30).

Now, it's the time to write code for the **DemoBean.java** file. Listing C.9 shows the code of the **DemoBean** Java class:

Listing C.9: Showing the Code of the DemoBean.java File

```
package com.kogent;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean(name="DemoBean")
@RequestScoped
public class DemoBean {
    private String name;
    public DemoBean() {
    }
    public String sayDemo() {
        return "hi";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```


In Listing C.9, the `@ManagedBean` annotation is used to specify that the class is a managed bean and the name attribute specify the name of the managed bean. The class has the name attribute, whose type is `String` and the `sayDemo()` method.

Creating the `faces-config.xml` File

You can create the JSF configuration file, which is a `faces-config.xml` file, by performing the following steps:

1. Right click the project node and select the `New`→`Other`→`JavaServer Faces`→`JSF Faces Configuration` option. The New JSF Faces Configuration wizard appears.

Figure C.31 shows the New JSF Faces Configuration wizard:

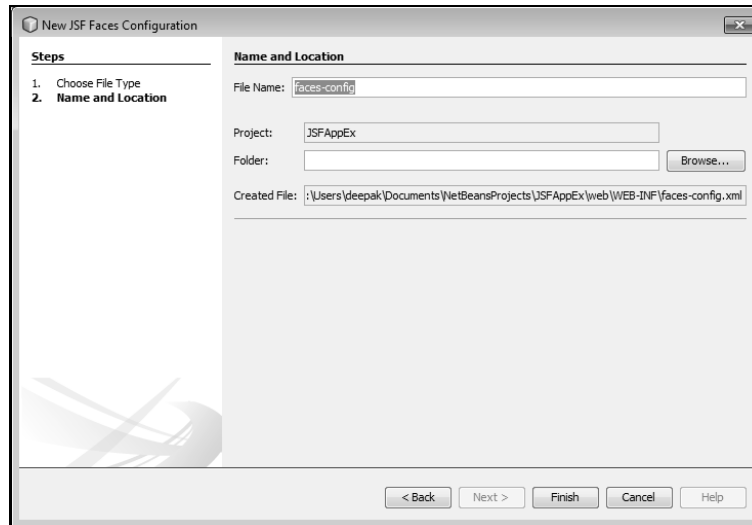


Figure C.31: Showing the New JSF Faces Configuration wizard

2. Click the `Finish` button (Figure C.31).

The `faces-config.xml` file appears in the Projects window.

Listing C.10 shows the code of the `faces-config.xml` JSF configuration file:

Listing C.10: Showing the Code of the `faces-config.xml` File

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
  <application>
    <resource-bundle>
      <base-name>com.kogent.messages</base-name>
      <var>msgs</var>
    </resource-bundle>
  </application>
  <navigation-rule>
    <from-view-id>/DemoMain.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>hi</from-outcome>
      <to-view-id>/DemoHello.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

In Listing C.10, the `<base-name>` tag, which is a sub-element of the `<resource-bundle>` tag, is used to specify the location of the `messages.properties` file. The navigation rule is specified using the `<navigation-rule>` tag, according to which navigation occurs in the application. The `<from-view-id>` tag specifies the view page from where navigation will start. The `<from-outcome>` tag specifies the String, which determines the view that is presented to the user. The `<to-view-id>` tag specifies the view that is presented to the user.

In the JSFAppEx application, when the user enters his/her name in the DemoMain XHTML page and submits, the `sayDemo()` method of the DemoBean class is invoked. As a result of the `sayDemo()` method invocation, the String value, `hi`, is returned. According to the rule specified in the `faces-config.xml` file, the DemoHello page is displayed to the user.

Building and Running the JSF Application

To build the JSFAppEx project, select the **Run** → **Build Project (JSFAppEx)** option. After the project is built successfully, you can run the application to view its output. Right click the project node and select the **Run** option to run the JSFAppEx application. On execution of the JSFAppEx application, the DemoMain page is displayed to the user, that is, the welcome page. Figure C.32 shows the output of the DemoMain page:



Figure C.32: Showing the Output of the DemoMain.xhtml File

By entering the name and clicking the Press button, the DemoHello XHTML page is displayed to the user.

Figure C.33 shows the output of the DemoHello page:

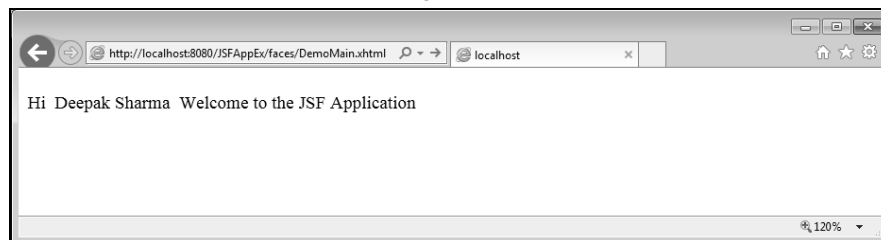


Figure C.33: Showing the Output of the DemoHello.xhtml File

This appendix has helped you to learn about the new features of the NetBeans IDE 7.4. You have also learned to develop Web application, enterprise application, and JSF applications using NetBeans IDE 7.4.