Developing Web Services Using Java Technology

Additional Material

DWS-4050-EE6 Rev A

D65185GC11
Edition 1.1
September 2010
D69081

ORACLE

Disclaimer

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Sun Microsystems, Inc. Disclaimer

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided. Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Book

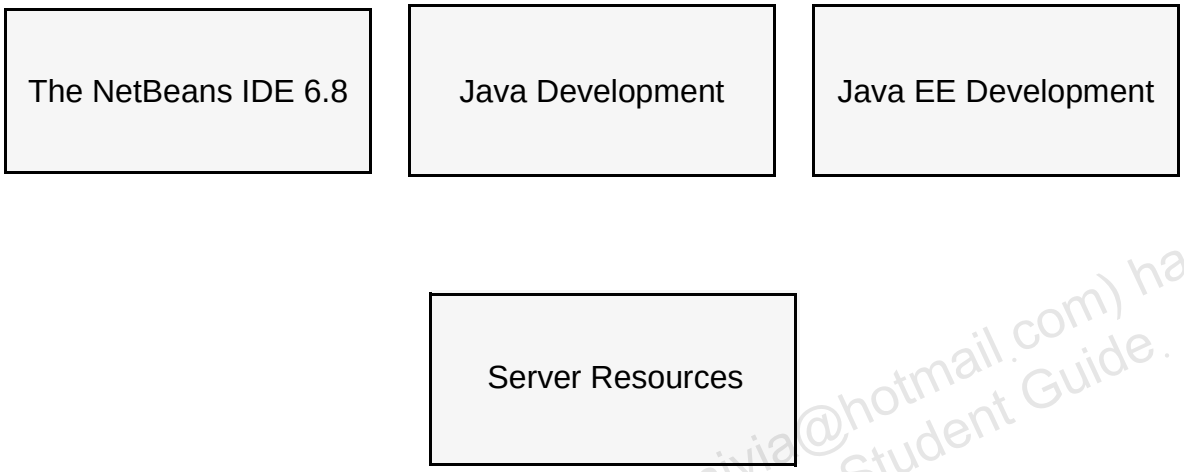
Goals

Upon completion of this book, you should be able to:

- Describe the NetBeans Integrated Development Environment (IDE)
- Use the NetBeans IDE for Java technology development
- Use the NetBeans IDE with the Java Platform, Enterprise Edition 6 (Java EE 6)
- View server resources using the NetBeans IDE

Book Map

The following book map enables you to see what you have accomplished and where you are going in reference to the book goals.



Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

> Use `ls -al` to list all files.
> `system% You have mail`.

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

> The `getServletInfo` method is used to get author information.
> The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

> To list the files in this directory, type:
> `# ls`

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

> Notice the `javax.servlet` interface is imported to allow access to its life cycle methods (Line 2).

`Courier italics` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

> To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

> Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for *`filename`* to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

> Read Chapter 6 in the *User's Guide*.
> These are called *class* options.

Additional Conventions

Java programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris Operating System is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris Operating System:

  ```
  $CD SERVER_ROOT/BIN
  ```

  If working in Microsoft Windows:

  ```
  C:\>CD SERVER_ROOT\BIN
  ```

Module 1

The NetBeans IDE 6.8

Objectives

Upon completion of this module, you should be able to:

- Understand the purpose of the NetBeans IDE
- Launch the NetBeans IDE
- Shut down the NetBeans IDE resources
- Set a larger font size for the IDE
- Add new or updated plugins to the IDE through the Plugins manager

The NetBeans IDE 6.8 Release

NetBeans IDE is an integrated development environment for creating, debugging, and deploying applications developed in many different programming languages, primarily Java.

The NetBeans IDE 6.8 release introduces comprehensive support for creating applications based on the Java EE 6 platform.

Downloading the NetBeans IDE

You can download the NetBeans IDE from the following URL:

```
http://www.netbeans.org/downloads/index.html
```

On this site, you can find downloads for specific operating systems and bundled distributions which include the following supported technologies:

- Java SE – Basic IDE that supports Java SE.
- JavaFX – JavaFX SDK bundle.
- Java – Java SE, Web and Java EE, Java ME, Groovy and bundled application servers.
- Ruby – Rails framework and Ruby included.
- C++ – Support for C, C++ and Fortran.
- PHP – Basic IDE with PHP support.
- All – Includes all of the above supported technologies in one download.

Additional plugins and NetBeans Platform support can be installed using the Tools – Plugins menu after installing the IDE.

NetBeans IDE Resources

You can find online information at the following URLs:

- `http://www.netbeans.org` – The NetBeans home page.
- `http://www.netbeans.org/community/releases/68/relnotes.html#system_requirements` – Minimum and recommended system requirements.
- `http://www.netbeans.org/community/releases/68/install.html` – Installation instructions. Covers all platforms and bundles.
- `http://wiki.netbeans.org/NetBeansUserFAQ` – The FAQ page. Answers many of the most frequent setup, configuration, and usage questions. Also has information on performance tuning.
- `http://www.netbeans.org/kb/docs/java/quickstart.html` – A brief tutorial that covers the creation and running of a `HelloWorld` project.
- `http://www.netbeans.org/kb/index.html` – The main documents page, including a list of learning trails, each of which contains multiple tutorials and other resources that guide you through the development of different types of applications.
- `http://www.netbeans.org/kb/kb.html` – Additional articles and tutorials, organized by topic.
- `http://www.netbeans.org/kb/docs/intro-screencasts.html` – Demos and screencasts of different IDE features.
- `http://www.netbeans.org/about/press/reviews.html` – Reviews of the NetBeans IDE.
- `http://planetnetbeans.org` – Blogs about NetBeans.
- `http://www.netbeans.org/community/lists/top.html` – Information about the NetBeans mailing lists.

Help System

The NetBeans IDE ships with an extensive help system. You can access the help system by selecting **`Help Contents`** from the Help menu, or by clicking the Help button in a wizard or dialog window.

Launching the NetBeans IDE

The NetBeans IDE is a Java application that is shipped with convenience scripts for launching it on the UNIX® platform and Microsoft Windows.

- On the Solaris Operating System (Solaris OS), type:

  ***$NETBEANS_HOME*/bin/netbeans**

- On the Microsoft Windows platform, type:

  ***%NETBEANS_HOME%*\bin\netbeans.exe**

If you install the NetBeans IDE on Windows using the installer, the installation process creates icons on the desktop and Start menu, making it convenient for launching the IDE. On UNIX, you must ensure that the *$NETBEANS_HOME*/bin directory is on your PATH environment variable to launch the IDE from the command line by typing the `netbeans` command.

The first time you launch the NetBeans IDE, it creates a `.netbeans\6.8` directory in the user's home directory (under `Documents and Settings\Username` on Microsoft Windows). This directory is known as the user directory. It is under this directory that user-specific settings are created. Subsequent launches of the NetBeans IDE read in these settings.

Shutting Down the NetBeans IDE

When you exit the NetBeans IDE, the current state of the IDE is saved and is restored the next time you start the IDE. The current state includes any projects that are open and the documents within those projects that are open. The NetBeans IDE prompts you to save any unsaved files before it shuts down.

Increasing the Font Size of the IDE

If the IDE's font sizes do not suit you, you can modify the IDE's *`netbeans.conf`* file to have the IDE use a different size.

To change the font size of the IDE, perform the following steps:

1. On your system, navigate to the IDE's *`$NETBEANS_HOME`*`/etc/netbeans.conf` file and open that file in a text editor.
2. On the line that begins `netbeans_default_options=` , insert the `--fontsize` switch and specify the font size that you want to use.

   For example, after you have finished, the first part of that line might look like the following:

   ```
   netbeans_default_options="--fontsize 18 -J-Xms32m
   ```
3. Restart the IDE to see the changes take effect.

Using the Plugin Manager of the IDE

The IDE's Plugin manager allows you to update your IDE dynamically. When you use the Plugin manager to update the IDE, the IDE checks the registered update centers to see if there are new plugins or new versions of already installed plugins available. If new or updated plugins are available, you can select, download, and install the plugins using the Plugin manager.

To add plugins from the Update Center:

1. Choose **`Tools > Plugins`** from the main menu to open the Plugin manager.

2. Click the **`Available Plugins`** tab to display plugins that are available for installation.
3. In the left pane, select the plugins you want to add by clicking the check box in the Install column. When you select a plugin, the version information and plugin description is displayed for the selected plugin in the right pane.
4. Click the **`Install`** button to install the selected plugins.
5. Complete the pages of the NetBeans IDE Installer wizard, including the accepting of licenses, to install the plugins.

Note – Some plugins may require you to restart the IDE to complete the update process.

Module 2

Java Development

Objectives

Upon completion of this module, you should be able to:

- Describe the Java application projects supported by the NetBeans IDE
- Use Java packages to organize the Java sources
- Create Java classes
- Use XML files
- Use other files

Java Application Projects

The main purpose of the NetBeans IDE is to make your Java development efforts easier and more efficient.

The NetBeans IDE organizes the source files for software development into projects. A project is a group of source files and the settings with which you build, run, and debug those source files. The IDE builds its project infrastructure directly on top of Apache Ant, the leading build tool for Java applications.

Project Types

The Java Platform, Standard Edition (Java SE) project types supported by the NetBeans IDE include:

- Java Application – A basic Java SE project with a main class.
- Java Class Library – A Java SE project without a main class.
- Java Project with Existing Sources – A Java SE project based on a set of existing source code files.
- Java Project with Existing Ant Script – A free-form Java SE project based on your existing Ant script.
- NetBeans Platform Application – A Java SE project based on the NetBeans Platform, which is a modular framework for Swing applications, providing a wide variety of out-of-the-box features to simplify the development and scalability of primarily desktop applications.

When creating a Java SE project, you usually use Java Application or Java Class Library templates. When you want to create more robust applications out of modules, without needing to start development from scratch, you use the NetBeans Platform.

Creating Projects

To create a new project, perform the following steps:

1. Select **`New Project`** from the File menu.
2. In the New Project wizard, choose the **`Java`** category and a project type.

3. Then click the **Next** button. The next screen you see depends upon the project type selected. Typically, you provide the project name and location for the project.
4. Click the **Finish** button.

Figure 2-1 illustrates the directory structure created with a Java Application project. You can view this directory structure in the Files window.

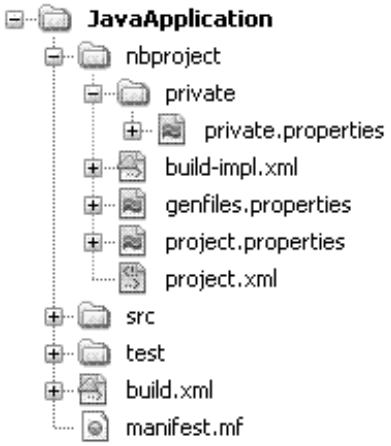


Figure 2-1 The Project Directory Structure

Opening Projects

To open an existing project, perform the following steps:

1. Select **Open Project** from the File menu.
2. Navigate to the location that contains the project folder. Depending on the type of project, a specific icon is displayed. For example, Figure 2-2 illustrates the icon shown for Java SE projects.
3. Select the project in the file dialog.
4. Check the Open as Main Project checkbox if you want the IDE to treat the project as a main project.
5. Check the Open Required Projects checkbox if you want to open associated projects in the IDE.
6. Click the **Open Project** button to open the project in the IDE.

Figure 2-2 The Icon for a NetBeans Java SE Project

Building Projects

To build a project, perform the following steps:

1. Switch to the Projects window.
2. Right-click your project node and select **Build** from the contextual menu.

Cleaning Projects

Cleaning a project removes any compiled class files from the project. To clean a project, perform the following steps:

1. Switch to the Projects window.
2. Right-click the project node in the Projects window and select **Clean**.

The `build` and `dist` directories are deleted by the IDE.

Running Projects

You can execute Java Application projects by right-clicking the project node in the Projects window and selecting **Run** from the menu. When you do this, any unsaved files in the project are saved and any uncompiled Java files are compiled. The class defined as the main class for the project is executed. If a main class for the project has not been configured, the IDE prompts the user to choose a main class for the project. Any classes that contain a static main method can be used for the main class for a project:

```
public static void main(String [] args)
```

Figure 2-3 illustrates the dialog box presented to users attempting to run a project that does not have a main class configured.

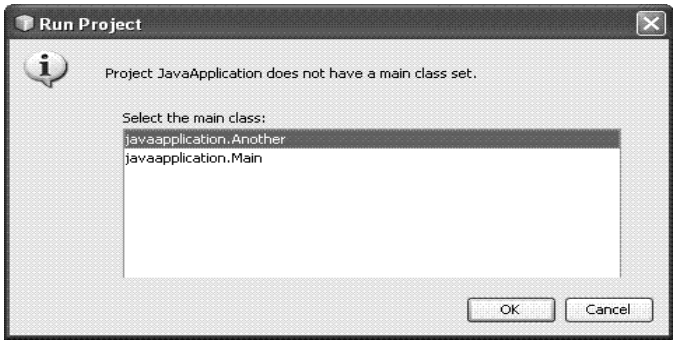


Figure 2-3 Choosing a Main Class to Run

Main Project

The NetBeans IDE allows you to have a single main project specified in the Projects window. The main project is the project that is operated on when you select *build main project* or *run main project*. You can build, compile, or execute other projects by right-clicking on the project and selecting build or run from the contextual menu.

To switch to the main project within the IDE, perform the following steps:

1. Switch to the Projects window.
2. Right-click the desired project node in the Projects window and select **`Set as Main Project`** from the contextual menu.

The selected project is now the main project.

Project Source Folders

Source files are organized into folders in a project. By default, every project is created with a directory called `src` for source packages and a directory called `test` for test packages. You can reconfigure these using the project properties window. To do this, perform the following steps:

1. Switch to the Projects window.
2. Right-click the project node and select **`Properties`** from the contextual menu.
3. Select **`Sources`** in the Categories pane.
4. In the Source Package Folders pane, you can use the following buttons:
   - Add Folder – Click this button to open a file dialog box that lets you specify a folder to add. Select the folder from the file system and click the **`Open`** button to add the folder.
   - Remove – Click this button to remove the selected folder from the list.
   - Move Up/Move Down– Click these buttons to change the positions of folders in the list.
5. Click the **`OK`** button to save your changes.

Modifying Project Libraries

In the Projects window, you can configure a project's classpath through the project's Libraries node. A project's libraries can include JAR files from another project, preconfigured libraries, or standalone JAR files or folders from the file system. To modify the libraries for a project, perform the following steps:

1. In the Projects window, expand your project.
2. Right-click the Libraries node under the project and select one of the menu items:
   - **`Add Project`** – Adds the JAR file from another project's `dist` directory to the current project's libraries. Navigate to the folder that contains the project folder, select the project folder, and click the **`Add Project JAR Files`** button.
   - **`Add Library`** – Adds a library from the Library Manager to the current project's libraries. Select the library from the list in the dialog box and click the **`Add Library`** button.
   - **`Add JAR/Folder`** – Adds the JAR file or folder from the file system to the current project's libraries.

You can also right-click a project node (or a project's Library node) and select **`Properties`**. Select the Libraries node in the dialog box. Figure 2-4 illustrates the project properties dialog window

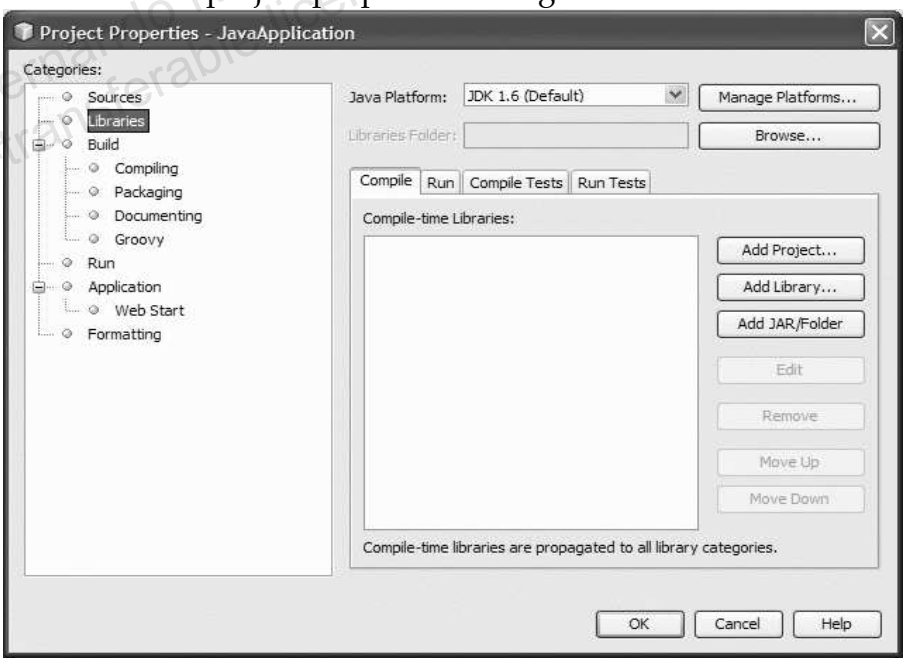


Figure 2-4 The Project Properties Dialog Window

In this window, you can configure the libraries (projects, preconfigured libraries, JAR files, and folders) that are used to compile, run, compile the tests, and run the tests for a project. Any libraries configured for the Compile tab are implicitly propagated to all other tabs.

Closing a Project

To close a project, select it in the Projects window and choose **Close *ProjectName*** from the File menu. Alternatively, you can right-click a project in the Projects window and select **Close**. Closing a project removes the project from the IDE's windows, but does not remove the project from the file system.

Deleting a Project

To delete a project, right-click the project node in the Projects window and select **Delete** from the menu. You are asked to confirm the deletion of the project and whether the IDE should delete the sources under the project directory. If you choose not to delete the sources, the project directory is left intact, but the NetBeans IDE configuration files (`nbproject` directory, `build.xml` Ant script) are removed.

Adding Ant Targets to a Project

You can edit the `build.xml` file to add Ant targets to a project. This file is not visible in the Projects window. You must use the Files window to see this file. Double-click the `build.xml` file to open it in the XML editor.

You can define new targets or override existing targets. The existing targets are in the `build-impl.xml` file in the `nbproject` directory. You should not edit the `build-impl.xml` file.

The `build-impl.xml` file defines several convenience targets that you can override to provide custom functionality. Table 2-1 illustrates some of these targets and when they are executed.

Table 2-1 Some Existing Ant Targets That You Can Override

Target	When Executed
`-pre-init`	Called before initialization of project properties
`-post-init`	Called after initialization of project properties
`-pre-compile`	Called before `javac` compilation
`-post-compile`	Called after `javac` compilation
`-pre-jar`	Called before building the JAR file
`-post-jar`	Called after building the JAR file
`-post-clean`	Called after cleaning the project

For example, if you wanted to execute an operation at the end of the build operation for a project, you could create a target in the `build.xml` file called `-post-jar`. In that target, define the operations you want to execute after the project is built.

Executing Customized Ant Targets

To execute an Ant target, expand the `build.xml` node in the Files window, find the target you want to execute, right-click the target node, and select **`Run Target`** from the menu. The targets from both `build.xml` and `build-impl.xml` files are listed under the `build.xml` node in alphabetical order, making it easy to find the target in question.

Setting the Main Class

The main class for a Java Application project is executed when you right-click the project node and select **`Run`**. The main class is specified in the `project.properties` file as the value of the `main.class` property. If you run a project but no main class is specified, the NetBeans IDE prompts you to set the main class. To set the main class for a project, perform the following steps:

1. In the Projects window, right-click the project node and select **`Properties`**.

2. In the Project Properties window, select **Run** from the left pane.
3. In the right pane, type the fully qualified class in the Main Class text field. You can also click the **Browse** button to display a list of classes in the project that contain a `main` method. Choose the class from the list and click the **Select Main Class** button.
4. Click the **OK** button to save your changes.

Figure 2-5 illustrates the Run settings in the properties window of a Java Application project.

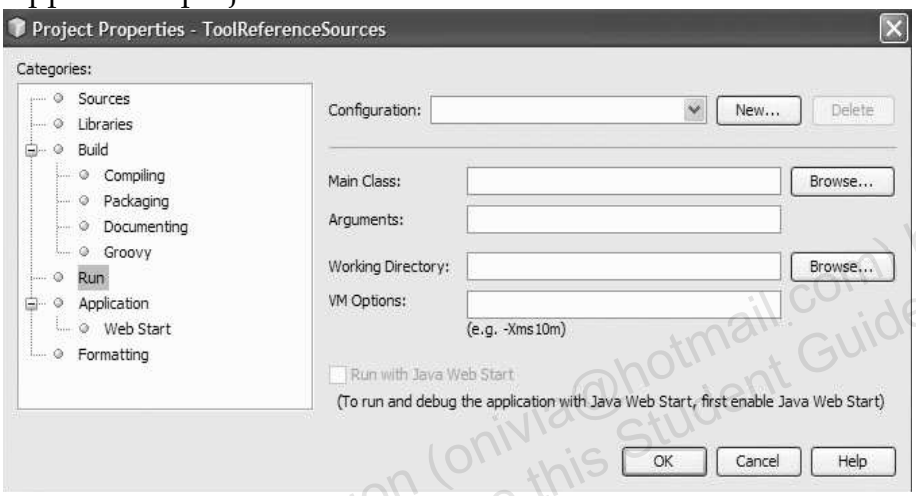


Figure 2-5 The Run Settings in the Project Properties Window

Setting Arguments

The arguments that are passed to the main class of a Java Application project are specified by the `application.args` property in the `private.properties` file. To set the arguments for a project, perform the following steps:

1. In the Projects window, right-click the project node and select **Properties**.
2. In the properties window, select **Run** from the left pane.
3. In the right pane, specify the arguments to be passed in the `Arguments` text field.
4. Click the **OK** button to save your changes.

Note – Arguments are only passed when you run the project (right-click the project node and select **Run**). If you right-click a class and select **Run File**, the application is run without arguments, even if arguments are set for the project.

Setting the Virtual Machine Options

The virtual machine (VM) options that are passed to all applications within a Java Application project are specified by the `run.jvmargs` property in the `project.properties` file. To set the VM options for a project, perform the following steps:

1. In the Projects window, right-click the project node and select **Properties**.
2. In the properties window, select **Run** from the left pane.
3. In the right pane, specify the options to be passed in the VM Options text field.
4. Click the **OK** button to save your changes.

Java Packages

Java packages are used to organize your Java sources. By default, packages are represented using dot notation in the Project window. To view packages in a tree structure, right-click the title of the Projects window and from the contextual menu, select **`View Java Packages as > Tree`**.

Creating Java Packages

To create a package, perform the following steps:

1. In the Projects window, right-click the project and select **`New > Java Package`**.

Note – The New submenu contains shortcuts to commonly-used templates and the Other command that you can use to open the New File wizard and access all NetBeans templates.

2. In the New Java Package wizard, you specify the following characteristics:
   - Package Name – The full package name (dot-separated package name).
   - Location – The location in the project for the package.
3. Click the **`Finish`** button.

The IDE creates the corresponding directory structure for you.

Modifying Java Packages

If you modify a Java package, this requires that any source code within that package hierarchy needs modification as well. Specifically, the `package` statement at the top of each file needs to change. The NetBeans IDE supports this refactoring.

Renaming Packages

To rename a package, perform the following steps:

1. In the Projects window, expand your project.

2. Expand the Source Packages node.
3. Right-click the package and select **`Refactor > Rename`** from the contextual menu.
4. In the Rename window, provide the following characteristics:
   - New Name – The new package name in this field.
   - Apply Rename on Comments – If you want the refactoring process to rename content in the comments, select this checkbox.
5. To complete the refactoring process without previewing changes, Click the **`Refactor`** button.

Note – To preview changes before they are made, you can click the Preview button. When you click the Preview button, the changes about to be made appear in the Refactoring window in the IDE. You can select the changes that you want to make and then click the Do Refactoring button to complete the refactoring process.

Deleting Packages

To delete a package, perform the following steps:

1. In the Projects window, expand your project.
2. Expand the Source Packages node.
3. Right-click the package node in the Projects window and select **`Delete`**.
4. You are prompted to confirm the deletion of the package. Click the **`Yes`** button to confirm deletion of the package.

The IDE deletes the package and its contents.

Compiling Java Packages

To compile a package, right-click the package node in the Projects window and select **`Compile Package`**. Alternatively, you can select the package and press the F9 key.

Java Classes

When creating a new Java class, you need a project in which you create the class. If you do not already have a project open, you need to either create or open a project.

Java Class Types

The NetBeans IDE has templates for creating the following Java class types:

- Plain Java Class – A class template that provides the package name, class definition, and a default constructor implementation.
- Empty Java File – No template is used for the file. The file created is entirely blank.
- Java Interface – An interface template that provides the package name and interface definition.
- Java Enum – An enum template that provides the package name and enum definition.
- Java Annotation Type – A template that provides a package name and `@interface` definition.
- Java Exception – A class template that provides a package name, class definition that extends from `java.lang.Exception`, a default constructor implementation, and a single-string constructor implementation.
- Java Main Class – A class template that provides a package name, class definition, default constructor implementation, and a static `main` method implementation.
- JApplet – A class template that provides a package name, class definition that extends from `javax.swing.JApplet`, and a default constructor.
- Applet – A class template that provides a package name, class definition that extends from `java.applet.Applet`, and an `init` method implementation.

JUnit Test Classes

The NetBeans IDE ships with JUnit support for unit testing your Java components. By default, JUnit test classes get created in the Test Packages area of a project.

Empty Test

When you create an empty test, you are prompted for the class name, location, package, and whether to implement `setUp` and `tearDown` methods. The template creates a file that imports the`junit.framework.*` file, defines a class that extends from `testCase`, and implements a constructor with a single string argument.

Creating Java Classes

To create a Java class, perform the following steps:

1. In the Projects window, right-click your project and select **`New > Other`**.

Note – Alternatively, you can select the project node and then select New File from the File menu.

---

2. In the New File dialog, choose **`Java`** from the Categories pane and the appropriate template from the File Types pane.
3. Click the **`Next`** button.
4. In the New Java Class dialog box, provide the following characteristics:
   - Class Name – The class name for the new class. Do not add the `.java` extension to the file name, because the IDE does this for you.
   - Location – Select the location within the project where the Java file is to be created.
   - Package – Select the package name, or type a new package name in this field.
5. Click the **`Finish`** button.

The IDE creates the class based on the selected template and opens the file in the editor.

Opening Java Classes

To open a Java class, perform the following steps:

1. In the Projects window, expand your project.
2. Expand the Source Packages node.
3. Expand the package node to locate the class in question.
4. Double-click the file node.

The IDE opens the class in the editor.

Modifying Java Classes

You can modify Java classes in many ways. The IDE provides support for modifying the structure (class name, methods, and fields) of all classes.

When editing a class, you can use the IDE's source editor to add fields, properties, methods, and static initializers.

Renaming Fields

The IDE enables you to easily rename a field and update references to that field accordingly. To rename a field, perform the following steps:

1. In the Projects window, expand your project.
2. Expand the Source Packages node.
3. Expand the package that contains your class.
4. Open the class file that you want to modify.
5. In the source editor, locate the field that you want to rename.

Note – To quickly locate the required field, in the Navigator window, right-click the field node and click Go to source.

6. In the source editor, right-click the field node and select **`Refactor > Rename`**.
7. In the Rename Field window, provide the following characteristics:
   - New Name – The new field name.

- Apply Rename on Comments – If you want the refactoring process to rename content in the comments, select this checkbox.
- To complete the refactoring process without previewing changes, Click the **`Refactor`** button.

Note – To preview changes before they are made, you can click the Preview button. When you click the Preview button, the changes about to be made appear in the Refactoring window in the IDE. You can select the changes that you want to make and then click the Do Refactoring button to complete the refactoring process.

Encapsulating Fields

The NetBeans IDE provides a refactoring mechanism to encapsulate fields. To use this functionality, perform the following steps:

1. In the Projects window, expand your project.
2. Expand the Source Packages node.
3. Expand the package that contains your class.
4. Right-click the class node and select **`Refactor > Encapsulate Fields`**.
5. In the resulting dialog box (see Figure 2-6 on page 2-17), choose the following characteristics:
   - For each field, use the checkboxes to specify the presence of `get` or `set` methods.
   - Fields' Visibility – Fields that have either `get` or `set` methods checked are rewritten to have the visibility specified here.
   - Accessor's Visibility – Fields that have either `get` or `set` methods checked have the accessor methods rewritten to have the visibility specified here.
   - Use Accessors Even When Field Is Accessible – If checked, access to the field within the class is rewritten to use the `get` or `set` methods.

6. To complete the refactoring process without previewing changes, Click the **Refactor** button.

Note – To preview changes before they are made, you can click the Preview button. When you click the Preview button, the changes about to be made appear in the Refactoring window in the IDE. You can select the changes that you want to make and then click the Do Refactoring button to complete the refactoring process.

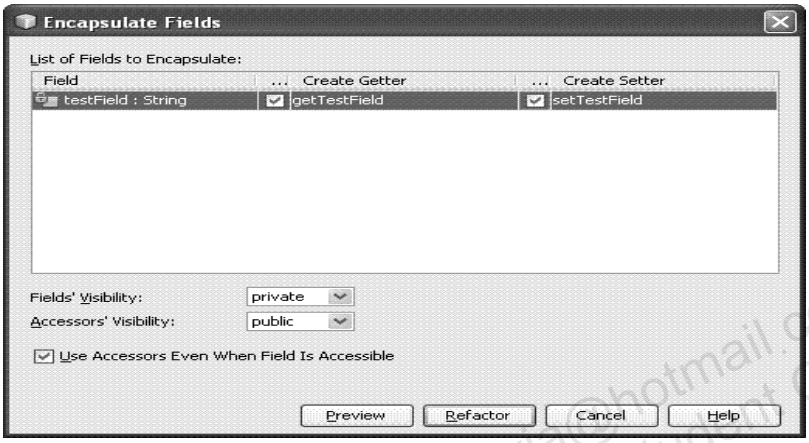


Figure 2-6 The Encapsulate Fields Dialog Box

Removing Fields

To remove a field in the IDE, perform the following steps:

1. In the Projects window, expand your project.
2. Expand the Source Packages node.
3. Expand the package that contains your class.
4. Open the class file that you want to modify.
5. In the source editor, locate the field that you want to delete.

Note – To quickly locate the required field, in the Navigator window, right-click the field node and click Go to source.

6. In the source editor, right-click the field name and select **Refactor > Safe Delete**.
7. Click the **Refactor** button to proceed. If the IDE does not detect usage of the field within the code, then it deletes the field. If the IDE does detect that the field is used, it provides a second confirmation window.

8. In the confirmation window, select from one of the following options:
   - To see where the field is used, click the **Show Usages** button.
   - To abort deletion, click the **Cancel** button.
   - To continue deletion, click the **Refactor** button.

Refactoring Methods

The NetBeans IDE supports refactoring of methods. When refactoring, the IDE attempts to identify uses of that method and modify them.

To rename a method, open the class that contains the method in the source editor and locate the method. Right-click the method name and select **Refactor > Rename**. Provide the new method name and click the **Refactor** button to complete the refactoring process without previewing changes. To preview changes before refactoring, click the **Preview** button. Next, preview the changes in the Refactoring window and click the **Do Refactoring** button.

To modify a method's parameters, open the class that contains the method in the source editor and locate the method. Right-click the method name and select **Refactor > Change Method Parameters**. Modify the parameters as required and click the **Preview** button. Review the changes in the Refactoring window, and then click the **Do Refactoring** button.

Removing Methods

To remove a method in the IDE, open the class that contains the method in the source editor and locate the method. Right-click the method name in the source editor and select **Refactor > Safe Delete**. Click the **Refactor** button to proceed. If the IDE does not detect usage of the method within the code, then it deletes the method. If the IDE does detect that the method is used, it provides a second confirmation window. To see where the method is used, click the **Show Usages** button. You can abort the delete by clicking the **Cancel** button. To continue the deletion, click the **Refactor** button.

Overriding Methods

When editing a class in the NetBeans IDE text editor, press **Control-space** to display a list of possible completions. You can then select the method you want to override. To override multiple methods, press **Alt-Insert** and then select **Override Method** from the pop-up menu to display the Generate Override Methods dialog box. Alternatively, you can select **Insert Code** from the Source menu, and then select **Override Method** from the pop-up menu. Select the method or methods you want to override. Click the **Generate** button to generate basic implementations of the methods. Figure 2-7 illustrates the methods inherited from `java.lang.Object` that are available for overriding.

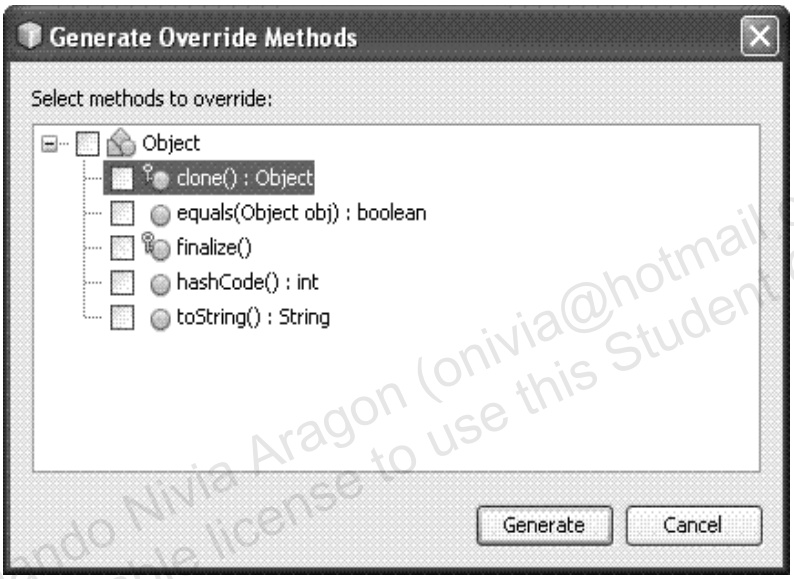


Figure 2-7 Methods Available for Overriding in a Class

Adding Constructors

To add a constructor to a class, in the source editor, press **Alt-Insert** and then select **Constructor** from the pop-up menu to display the Generate Constructor dialog box. Alternatively, you can select **Insert Code** from the Source menu, and then select **Constructor** from the pop-up menu. In the Generate Constructor dialog box, specify the parameters the constructor requires and the fields that you want to initialize. Click the **Generate** button to add the constructor.

Compiling Java Classes

To compile a Java class, right-click the file node in the Projects window and select **`Compile File`**. Alternatively, you can press **`F9`** when you select the file in the Projects window or have the file open in the IDE's text editor.

Executing Java Programs

The NetBeans IDE can execute Java programs. Output from programs is directed to an output window specific to the executing program.

Main Classes

You can execute any Java class that has defined a public static main method. To execute such a class, right-click the file in the Projects window and select **`Run File`**. Alternatively, you can press **`Shift-F6`** when you select the file in the Projects window or have the file open in the IDE's text editor.

JUnit Test Classes

To execute a JUnit test class, right-click the test in the Projects window and select **`Run File`**. Or, you can press **`Shift-F6`** when you select the file in the Projects window or have the file open in the IDE's text editor. When you run a test, the IDE displays output in two tabs. A summary of the passed and failed tests and the description of failed tests are displayed in the JUnit Test Results window. The Output window displays any information that the `test` methods write to `System.out`. Figure 2-8 illustrates the JUnit Test Results window.

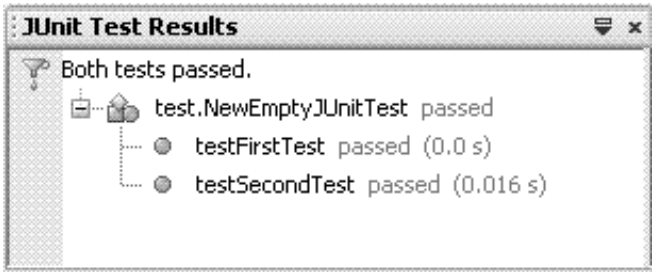


Figure 2-8 The JUnit Test Results Window

Copying Java Classes

Java classes can be copied from one project to another and even within a project. To copy a Java class, right-click the class and select **Copy**, then right-click the destination for the class and select **Paste > Copy** or **Paste > Refactor Copy**. When you select **Paste > Refactor Copy**, the Copy Class dialog box is displayed. Specify a new name for the class and click the **Preview** button. Review the changes in the Refactoring window, and then click the **Do Refactoring** button. When you use the IDE to copy a class, the `package` statement at the top of the Java file is modified to reflect the package into which the file was copied.

Moving Java Classes

You can move Java classes from one package to another by simply dragging and dropping the file. You can also use cut-and-paste: right-click the file and select **Cut**, then right-click the destination for the class and select **Paste > Move** or **Paste > Refactor Move**. When you select **Paste > Refactor Move**, the IDE presents you with the Move Class dialog box that allows you to specify whether the class should be moved without refactoring. You can click the **Preview** button to preview the changes in the Refactoring window, and then click the **Do Refactoring** button. When you use the IDE to move a class, the `package` statement at the top of the Java file is modified to reflect the new package name. Figure 2-9 illustrates the Move Class dialog box.

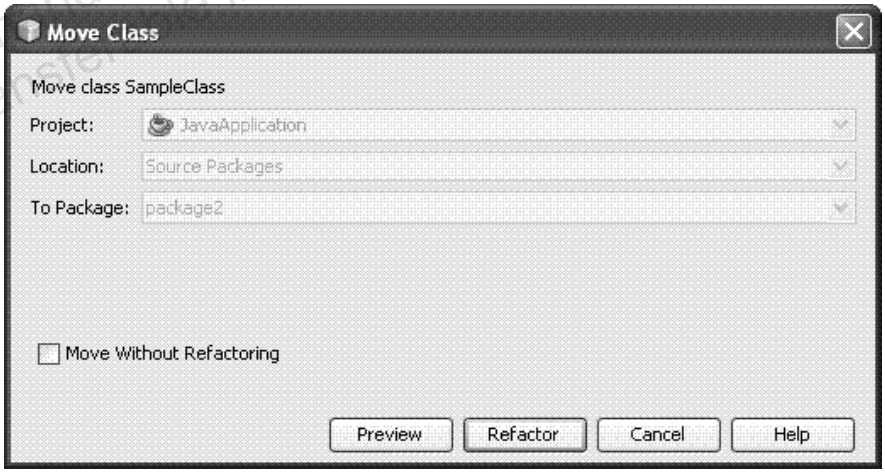


Figure 2-9 The Move Class Dialog

Copying Existing Resources

You can use the Favorites window to copy files to a project. If the Favorites window is not visible, choose **Favorites** from the Window menu. In the Favorites window, navigate to the file you want to copy into the project. Right-click the file and select **Copy** from the contextual menu. Switch back to the Projects window and navigate to the package that needs to contain the file. Right-click the package and select **Paste** from the contextual menu. You can add directories to the Favorites window by right-clicking in the Favorites window and selecting **Add to Favorites**. Navigate to the directory you want to add, then click the **Add** button.

If you are copying resources to replace existing files in a project, you should first delete those existing files. If you do not, the IDE appends an underscore and number to the end of the file name (and automatically refactor the class), which is probably not what you want.

Terminating a Running Process

If you want to terminate a Java process running in the NetBeans IDE, select **Processes** from the **Window** menu. The running processes are displayed in the lower-right coner of the IDE. Right-click the process you want to terminate and select **Cancel Process**. In the confirmation window, click **Yes** to terminate the running process.

XML Files

The NetBeans IDE provides an XML-aware editor that you can use to create and validate XML files.

Creating XML Files

To create a new XML document, right-click the project node and select **New > Other**. Choose **XML** from the Categories pane and **XML Document** from the File Types pane. Click the **Next** button. On the first screen of the New XML Document wizard, you provide the file name for the XML document and the folder in the project where the XML document will be placed. Then click the **Next** button. On the second screen of the wizard, you can select whether you are creating a well-formed XML document, or one that is constrained by a DTD file or XML schema.

If you choose a well-formed document, click the **Finish** button to create the XML document. The document opens in the NetBeans IDE XML editor.

If you choose a DTD-constrained document, click the **Next** button. On the next screen of the wizard, supply the DTD public ID, DTD system ID, and document root for the XML document. Click the **Finish** button to create the XML document. The document opens in the NetBeans IDE XML editor.

If you choose a schema-constrained document, click the **Next** button. On the next screen of the wizard, supply the schema URI, document namespace, and root element. Click the **Finish** button to create the XML document. The document opens in the NetBeans IDE XML editor.

Opening XML Files

Unless an XML document is stored under the `src` directory in a project, it does not appear in the Projects window. You can navigate to an XML file in the Files window. Double-click an XML file to open it.

Checking and Validating XML Files

The NetBeans IDE can check an XML document to see that it is well-formed. XML documents are considered well-formed if they adhere to XML grammatical rules. To check an XML document, right-click the XML document node in the Files window and select **Check XML**. The results of the XML check appear in the Output window.

The NetBeans IDE can validate an XML document that is constrained by a DTD or schema. To validate an XML document, right-click the XML document node in the Files window and select **Validate XML**. The results of the XML validation appear in the Output window.

Other Files

Some non-standard directories and rarely used directories do not appear in the Projects window. If you want to see any of these directories, such as your build script or files in the root directory of your project, use the Files window.

File Types

In the New File wizard, the Other entry in the Categories pane groups some file templates that do not fit into the other categories. You can use this to create the following types of files:

- Properties Files – Name/value pairs in a file that ends with the `.properties` extension
- Ant Build Script – An empty Ant build script
- Cascading Style Sheet – Creates an empty cascading style sheet (CSS) file
- Empty File – A text file with an arbitrary file extension
- Folder – A directory within the project
- HTML File – A basic file based on the HTML template, ending with the `.html` extension
- Java Script File – Creates an empty JavaScript file in the IDE's Source Editor
- JNLP File – Creates a Java Network Launching Protocol (JNLP) file formatted in XML
- JSON File – Creates empty JavaScript Object Notation (JSON) file
- SQL File – Creates an empty Structured Query Language (SQL) file
- SVG File – Creates empty Scalable Vector Graphics (SVG) Tiny 1.1 file
- XHTML File – An XHTML-strict file that can be validated against a Document Type Definition (DTD), ending with the `.html` extension
- YAML File - Creates an empty YAML file in the IDE's Source Editor

Creating New Files

To create a new file, right-click on the project node and select **New > Other**. Select **Other** from the Categories pane and the appropriate file type from the File Types pane. Then click the **Next** button. In the window that follows, provide the file name and location for the file. Do not include the extension in the file name unless you are creating an empty file. Click the **Finish** button to create the file. The new file opens in the NetBeans IDE text editor.

Opening Files

To open a file, locate the file node in the Files window and double-click the file node. The file opens in the NetBeans IDE text editor.

Creating Folders

To create a folder in a project, right-click the project node in the Files window and select **New > Other**. Select **Other** in the Categories pane and **Folder** in the File Types pane and click the **Next** button. In the next window, provide the folder name and parent folder location. Click the **Finish** button to complete the folder creation.

Deleting Files

To delete a file, right-click the file node in the Files window and select **Delete**. The NetBeans IDE prompts you to confirm the deletion of the file. When you confirm the deletion, the IDE deletes the file from the file system.

Copying Files and Folders

To copy a file or folder, locate the file or folder node in the Files window. Right-click the node and select **Copy**. Locate the destination for the copied resource. Right-click the node for the destination and select **Paste**.

Moving Files and Folders

To move a file or folder, locate the file or folder node in the Files window. Right-click the node and select **Cut**. Locate the destination for the moved resource. Right-click the node for the destination and select **Paste**.

Module 3

Java EE Development

Objectives

Upon completion of this module, you should be able to:

- Create web applications
- Create enterprise applications
- Create Enterprise JavaBeans components
- Create web services

Enterprise Application Projects

The NetBeans IDE provides full Java EE application support in the form of enterprise application projects. The IDE organizes the files that make an enterprise application

Creating Enterprise Application Projects

To create a new enterprise application:

1. Select **`New Project`** from the File menu.
2. Select **`Java EE`** from the Categories pane.
3. Select **`Enterprise Application`** from the Projects pane and click the **`Next`** button.
4. Provide the following information in the next window:
   - Project Name – Used as the base name for the directory as well as the base name for the sub projects (if selected)
   - Project Location – Used to specify the location for the base project directory
   - Use Dedicated Folder for Storing Libraries – Used to specify a Libraries Folder where required compilation libraries are stored.
   - Set as Main Project – If checked, the enterprise application project is set as the main project within the IDE.
5. Click Next. In the next window, provide the following information:
   - Server – Specifies the server to be used for deployment
   - J2EE Version – Allows you to select either Java EE 6, Java EE 5 or J2EE 1.4. If you select J2EE 1.4, an additional checkbox appears that allows you to set the source level to J2SE 1.4
   - Create EJB Module – If checked, you can supply the name for the EJB module project. The project directory is created within the Enterprise Application project directory.
   - Create Web Application Module – if checked, you can supply the name for the web application module project. The project directory will be created within the Enterprise Application project directory.

- Create Application Client Module – If checked, you can supply the name for the Java EE client application module project. The project directory is created within the Enterprise Application project directory.

6. Click the **Finish** button to create the enterprise application.

The NetBeans IDE lists the enterprise application project in the Projects window. Any sub-projects (the web application module or EJB module) are also listed as projects in the Projects window.

Figure 3-1 illustrates the directory structure created for an enterprise application that contains a web application module and EJB module. This can be seen in the Files window. The project directories for the modules are under the directory for the enterprise application project.

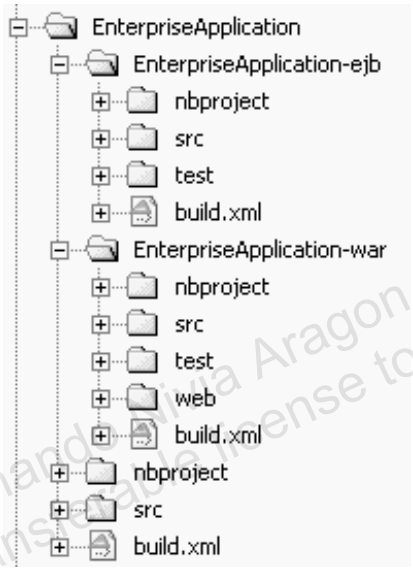


Figure 3-1 The Enterprise Application Project Structure

When you want to add a specific Java EE component, you need to do that in the appropriate project. For example, you must create a servlet class within a web application module project.

Creating Deployment Descriptors

Java EE application deployment descriptors are optional for Java EE version 5 onwards. The following sections provide instructions for creating application deployment descriptors using the NetBeans IDE.

Creating the Standard Application Deployment Descriptor

To create the standard Java EE application deployment descriptor (`application.xml` file), perform the following steps:

1. Right-click your Enterprise Application project.
2. Select **`New`** > **`Standard Deployment Descriptor`** menu option.
3. Click the **`Finish`** button to create the `application.xml` file.
4. The deployment descriptor opens in the NetBeans IDE XML editor. You must type the raw XML for any entries you want in this descriptor.

Creating the GlassFish Server Specific Application Deployment Descriptor

To create the GlassFish Server specific application deployment descriptor (`sun-application.xml` file), perform the following steps:

1. Right-click your Enterprise Application project.
2. Select **`New`** > **`Other`** menu option.
3. Select **`GlassFish`** from the Categories pane.
4. Select **`GlassFish Deployment Descriptor`** from the File Types pane and click the **`Next`** button.
5. Click the **`Finish`** button to create the `sun-application.xml` file.
6. The deployment descriptor opens in the NetBeans IDE XML editor.
7. The sun-application deployment descriptor opens in the visual editor. This editor allows you to edit some of the GlassFish server specific settings for the Java EE application.

Note – If the `sun-application.xml` file is already open in the visual editor, you can open the file in the XML editor by clicking the XML View button.

Configuring Deployment Descriptors

In the NetBeans IDE, you can find the deployment descriptors for an enterprise application in the Projects window under the Configuration Files node of the enterprise application project.

Editing the Standard Application Deployment Descriptor

To edit the standard Java EE application deployment descriptor, perform the following steps:

1. Expand the Configuration Files node in your Enterprise Application project.
2. Double-click the `application.xml` file node (or right-click the node and select **`Edit`**).
3. The deployment descriptor opens in the NetBeans IDE XML editor. You must type the raw XML for any entries you want in this descriptor.

Editing the GlassFish Server Application Deployment Descriptor

To edit the GlassFish Server application deployment descriptor, perform the following steps:

1. Expand the Configuration Files node in your Enterprise Application project.
2. Double-click the `sun-application.xml` file node (or right-click the node and select **`Open`**).
3. The Sun-specific deployment descriptor opens in the visual editor. This editor allows you to edit some of the GlassFish Server specific settings for the Java EE application.

Note – If the `sun-application.xml` file is already open in the visual editor, you can open the file in the XML editor by clicking the XML View button.

Adding Build Packages

Build packages can be added to an enterprise application project. These build packages will be packaged as part of the EAR file that is generated. To add build packages, perform the following steps:

1. Right-click the enterprise application project node and select **`Properties`**.
2. In the project properties window, select **`Build > Packaging`** in the left pane.

3. In the right pane, use the **Add** buttons to add the appropriate content.

For example, to add a project, click the **Add Project** button. In the Select Project window, navigate to the project you want to add, select it, and click the **Add Project JAR Files** button. The project JAR is added to the enterprise application project. The Location In Archive column determines the directory in the EAR file where the JAR file will be placed. The root directory (/) is the default location.

Building Java EE Applications

To build a Java EE application, right-click the enterprise application project node and select **Build**. The IDE performs the following:

- For each web application module that is part of the enterprise application:
  - Compiles the Java source code into the `build` directory of the web application project
  - Creates a WAR file in the `dist` subdirectory of the web application project
- For each EJB module that is part of the enterprise application:
  - Compiles the Java source code into the `build` directory of the EJB module
  - Creates an EJB JAR file in the `dist` directory of the EJB module project
- Copies each WAR and EJB JAR file to the `build` directory of the enterprise application project
- Creates an EAR file in the `dist` directory of the enterprise application project

When you build a Java EE application, any open files are first saved, then the compilation and assembly process begins.

Deploying Java EE Applications

To deploy a Java EE application from the NetBeans IDE:

1. Right-click the enterprise application node and select **Deploy**.

2. The IDE deploys the EAR file to the application server. If the application has not been built, or if it has changed since the last time it was built, the IDE re-builds the application before deploying the EAR file.

The EAR file is deployed to the application server configured as the server in the enterprise application project's properties.

Undeploying Java EE Applications

To undeploy a Java EE application from the application server:

1. Change to the Services window.
2. Expand the Servers node.
3. Expand the GlassFish node.
4. Expand the Applications node.
5. Right-click the application you want to remove and select **Undeploy**.

The Output window displays the undeployment progress.

Configuring Java EE Resources

You can use the NetBeans IDE to configure Java EE resources in the application server.

Configuring Connection Pools

Connection pools are supported by web application projects, EJB modules, and enterprise application projects.

Creating a Connection Pool Resource

To create a connection pool resource:

1. Expand your application project.
2. Right-click the Server Resources node and select **`New > Other`**.
3. Select **`GlassFish`** from the Categories pane and **`JDBC Connection Pool`** from the File Types pane.
4. Click the **`Next`** button to continue.
5. Supply the name for the connection pool.
6. Choose the appropriate radio button to either:
   a. Create the connection pool by extracting information from an existing connection in the IDE. Choose the appropriate connection from the drop-down list.
   b. Create a new configuration by specifying the Database vendor. Choose Java DB (Net) from the drop-down list for the Java DB database.
7. Check the XA (Global Transaction) checkbox if you want the connection pool to support global transactions.
8. Click the **`Next`** button to continue.
9. On the next screen, provide the data source class name. For the Java DB database, it should be:

   `org.apache.derby.jdbc.ClientDataSource`

   The value for resource type should be:

   `java.sql.DataSource`

10. In the properties section, provide the connection properties for the database connection pool. For the Java DB database, you must supply the following properties:
    - `serverName`
    - `PortNumber`
    - `DatabaseName`
    - `User`
    - `Password`
11. Use the **Add** button to add property lines to the table. To delete a property, click once on the property to select it and click the **Remove** button.
12. You must provide values for any defined properties to continue.
13. Click the **Finish** button to create the resource within your project.
14. If you click the **Next** button, you are presented with a final screen that can be used to configure additional connection pool properties.

When created, the connection pool resource details are added to the `sun-resources.xml` descriptor file under the Server Resources node in your project. The IDE will create the `sun-resources.xml` descriptor file if the file does not exist.

Configuring JDBC Resources

JDBC JNDI resources are supported by web application projects, EJB modules, and enterprise application projects.

Creating a JDBC Resource

To create a JDBC resource, perform the following steps:

1. Expand your application project.
2. Right-click the Server Resources node and select **New > Other**.
3. Select **GlassFish** from the Categories pane and **JDBC Resource** from the File Types pane.
4. Click the **Next** button to continue.
5. In the next window of the wizard, you can choose to either:
    a. Create the resource with an existing JDBC connection pool.

b. Create the resource with a new JDBC connection pool.

6. Provide the JNDI name, object type, enabled status (true or false) and optional description for the JDBC resource.
7. If you chose to create the resource using an existing JDBC connection pool, click the **Finish** button.
8. If you chose to create a new JDBC connection pool:
   a. Click the **Next** button.
   b. On the next screen, provide any additional properties for the JDBC resource.
   c. Click the **Next** button to continue.
   d. The wizard steps you through configuring the connection pool. See "Configuring Connection Pools" on page 3-8 for more information.

When created, the JDBC resource details are added to the `sun-resources.xml` descriptor file under the Server Resources node in your project. The IDE will create the `sun-resources.xml` descriptor file if the file does not exist.

Configuring JMS Resources

You can use the IDE to create JMS Administration Object Resources and JMS Connector Resources in your EJB module project. JMS resources are registered with the GlassFish Server when the project is deployed. You can also manually register the resources with the server in the Projects window.

Creating a JMS Resource

To create a JMS resource, perform the following steps:

1. In the Projects window of the IDE, right-click the EJB module project node and select **New > Other**.
2. Select **GlassFish** from the Categories pane and **JMS Resource** from the File Types pane.
3. Click the Next button to continue.
4. Provide the JNDI name, enabled status (true or false) and optional description for the JMS resource.
5. Choose the resource type you want to create.

6. Click the Next button to add any additional properties.
7. Click the Finish button.

When created, the JMS resource details are added to the `sun-resources.xml` descriptor file under the Server Resources node in your project. The IDE will create the `sun-resources.xml` descriptor file if the file does not exist.

Deleting a JMS Resource from the EJB Module

To delete a JMS resource from your project, perform the following steps:

1. In the Projects window of the IDE, expand your EJB module project.
2. Expand the Server Resources node.
3. Double-click the `sun-resources.xml` descriptor file to open the file in the XMl editor
4. Edit the XML file to remove the JMS elements. Save your changes.

Removing a JMS Resource From the GlassFish Server

To remove a JMS resource from the GlassFish Server, perform the following steps.

1. In the Services window of the IDE, expand the GlassFish Server node.
2. Expand the Connectors node under the Resources node.
3. Locate the JMS resource you want to delete by expanding the Connector Resources node, the Connector Connection Pools node, and the Admin Object Resources node.
4. Right-click each of the resource nodes that you want to delete and choose Unregister. The Output window displays the status of unregistering the resource. In the Services window, the node for the deleted resource is removed.

Note – To remove a JMS resource from your project, you should remove the JMS resource from the server and also delete the resource from the EJB module project. If you do not delete the resource from the EJB module project, the resource will be registered again with the server when you deploy the application.

Web Applications

To create web components, you must first create a Web Application project (either standalone or as a module within an Enterprise Application project) in the NetBeans IDE.

Web Application Projects

A web application project encompasses all the components that are used in the web tier of a Java EE application. The NetBeans IDE organizes the source files for a web application, using Ant to build and deploy the artifacts (the WAR file). A web application project can be standalone (that is, it is not part of a larger Java EE application) or it can be one module in a larger enterprise application.

To create a new web application project, select **New Project** from the File menu. In the New Project wizard, choose **Web** from the Categories pane and the appropriate project type from the Projects pane. Possible project types are:

- Web Application – This produces a new web application.
- Web Application with Existing Sources – This produces a standard web application from an existing source tree.
- Web Application with Existing Ant Script – This produces a free-form web application using your existing Ant script.

Creating a Web Application Project

To create a new web application project, perform the following steps:

1. Select **New Project** from the File menu.
2. Choose **Web** from the Categories pane and **Web Application** from the **Projects** pane.
3. Click the **Next** button.
4. In the next window of the wizard, provide the following characteristics:
   - Project Name – The project name is used as the name for the project directory. For cross-platform compatibility, this name should not contain spaces.

- Project Location – The location used for the base directory for the project. The project folder will be based on the project name. The full path is displayed in the Project Folder field.
- Add to Enterprise Application – You can choose to add this web application to an existing enterprise application.
- Server – Choose the application server that is used for deployment of this application.
- Java EE Version – Choose the appropriate Java EE version for the web application (J2EE 1.3, J2EE 1.4 or Java EE 5).
- Context Path – This is the context path that is used for URLs to access this web application.
- Main Project – If checked, this project will be marked as the main project within the IDE.

5. Click the **Next** button to continue.
6. On the next screen you can select the framework (Spring Web MVC 2.5, JavaServer Faces, Struts 1.3.8, or Hibernate 3.2.5) you will be using with the web application. Choosing a framework adds the appropriate libraries and configuration files to the web application project.
7. Click the **Finish** button to create the web application project similar to one shown in Figure 3-2.

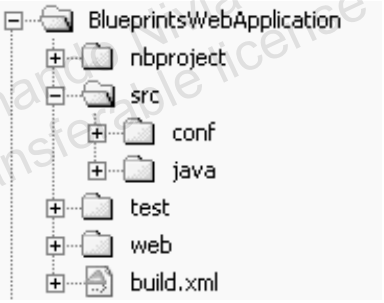


Figure 3-2 The Web Application Project Structure

Servlets

The NetBeans IDE provides a template for creating servlets.

Creating Servlets

To create a new servlet, perform the following steps:

1. Right-click the web application project node and select **New > Other**.
2. Select **Web** from the Categories pane and **Servlet** from the File Types pane.
3. Click the **Next** button to continue.
4. In the next window of the wizard, supply the following characteristics:
   - Class Name – Specify the name for the servlet class.
   - Location – Specify the location within the project for the servlet source code.
   - Package – Select or type the package name for the servlet class.
5. Click the **Next** button.
6. In the next window of the wizard, you can choose to add the servlet information to the deployment descriptor. In Java EE 6, you typically want to add servlet information as an annotation in the servlet class, rather than in the deployment descriptor. Therefore, you would leave this checkbox unselected and the IDE will add annotations to the servlet class for you.
   If, however, you select the **Add information to deployment descriptor** checkbox, you can provide the following characteristics for the deployment descriptor:
   - Servlet Name – The logical name for the servlet.
   - URL Pattern(s) – A comma-separated list of URL patterns for the servlet. These are placed into individual `servlet-mapping` elements in the deployment descriptor.
   - Initialization Parameters – To provide initialization parameters for the servlet, use the **New** button. These are added within the `servlet` element in the deployment descriptor.
7. Click the **Finish** button to create the servlet in the web application project.

The source code for the servlet opens in the NetBeans IDE code editor.

Note – The template for servlets is configured such that the `doGet` and `doPost` methods call a common method called `processRequest`. In the editor, the `HttpServlet` methods are collapsed at the bottom of the code. To see these, click the + symbol to expand that section of the code.

Deleting Servlets

To delete a servlet from the web application project, perform the following steps:

1. In the Projects window, expand the web application project.
2. Expand the Source Packages node.
3. Expand the appropriate package nodes until you see the file node for the servlet.
4. Right-click the file node for the servlet and select **`Delete`**.
5. The IDE asks you to confirm deletion of the class. Click the **`Yes`** button to delete the servlet file.
6. Expand the Configuration Files node for your project.
7. Double-click the `web.xml` node to open the deployment descriptor.
8. In the visual editor, select the **`Servlets`** button.
9. Select the servlet to be removed.
10. Click the **`Remove`** button.
11. The IDE asks you to confirm the deletion of the servlet from the deployment descriptor. Click the **`Yes`** button to delete the servlet entry from the deployment descriptor.

Creating JavaServer Pages

The NetBeans IDE provides a template for creating JavaServer Pages (JSP) pages. To create a new JSP page, perform the following:

1. In the Projects windows, right-click the web application project node and select **`New > Other`**.
2. Select **`Web`** from the Categories pane and **`JSP`** from the File Types pane.
3. Click the **`Next`** button to continue.
4. In the next window of the wizard, provide the following characteristics:
   - JSP File Name – The file name for the JSP file. Do not provide the `.jsp` extension, because that is added by the wizard.
   - Location – The location in the project where the JSP file will be created.
   - Folder – A subfolder for the JSP file, if necessary.

- Options – JSP File or JSP Document. JSP Files use standard HTML syntax, whereas JSP Documents must be written with XML syntax.
- Create as a JSP Segment – If you are creating a JSP segment, the file extension is `.jspf` instead of the standard `.jsp` extension.

5. Click the **`Finish`** button to create the JSP file in the web application project.

The JSP file opens in the NetBeans IDE JSP editor.

Editing JSP Pages

When editing a JSP page, you can add common components to the contents of the JSP file using the JSP palette window. Figure 3-3 illustrates this window. To open the palette window, select **`Palette`** from the Window menu.

Figure 3-3 The JSP Palette Window

There are two ways to add a component in the palette to your JSP page:

- Double-click – Position your cursor within the JSP file at the location you want the component added. Double-click the component in the palette.
- Drag-and-drop – Select the component in the palette, hold your mouse button down and drag the component to the desired location within the JSP file.

In either case, each component has properties that you need to set, and a dialog box reflecting those properties opens. Provide the information in the dialog box and click the **OK** button to add the component to the JSP file.

HTML Files

The NetBeans IDE provides a template for creating HTML files. To create a new HTML file, perform the following steps:

1. In the Projects window, right-click the web application project node and select **New > Other**.
2. Select **Web** from the Categories pane and **HTML File** from the File Types pane.
3. Click the **Next** button to continue.
4. In the next window of the wizard, provide the following characteristics:
   - File Name – The file name for the HTML file. Do not provide the `.html` extension, because that is added by the wizard.
   - Folder – A subfolder for the HTML file, if necessary.
5. Click the **Finish** button to create the HTML file in the web application project.

The HTML file opens in the NetBeans IDE HTML editor.

Web Application Listeners

The NetBeans IDE provides a template for creating the various listener classes supported by web applications.

Creating Listener Classes

To create a new listener class, perform the following steps:

1. In the Projects window, right-click the web application project node and select **New > Other**.
2. Select **Web** from the Categories pane and **Web Application Listener** from the File Types pane.
3. Click the **Next** button to continue.

4. In the next window of the wizard, provide the following characteristics:
   - Class Name – Specify the name for the listener class.
   - Location – Specify the location within the project for the listener source code.
   - Package – Select or type the package name for the listener class.
   - Interfaces – Check the interfaces that you want the listener class to implement.
   - Add information to deployment descriptor – If checked, the listener element is added to the deployment descriptor.
5. Click the **Finish** button to create the listener.

The listener class opens in the NetBeans IDE source editor.

Deleting Listener Classes

To delete a listener from the web application project, perform the following steps:

1. In the Projects window, expand the web application project.
2. Expand the Source Packages node.
3. Expand the appropriate package nodes until you see the file node for the listener class.
4. Right-click the file node for the listener and select **Delete**.
5. Expand the Configuration Files node for your project.
6. Double-click the `web.xml` node to open the deployment descriptor.
7. In the visual editor, select the **General** button.
8. Expand the **Web Application Listeners** node in the visual editor.
9. Select the listener to be removed.
10. Click the **Remove** button.

The listener entry is removed from the deployment descriptor.

Filter Classes

The NetBeans IDE provides a template for creating filters in your web application projects.

Creating Filters

To create a new filter class, perform the following steps:

1. In the Projects window, right-click the web application project node and select **`New > Other`**.
2. Select **`Web`** from the Categories pane and **`Filter`** from the File Types pane.
3. Click the **`Next`** button to continue.
4. In the next window of the wizard, provide the class name, location, and package for the filter. In this window, you can also choose to wrap the request and response objects. If you check this option, the IDE creates inner classes called `RequestWrapper` and `ResponseWrapper` that extend from `HttpServletRequestWrapper` and `HttpServletResponseWrapper` respectively.
5. Click the **`Next`** button to continue.
6. In the next window of the wizard, you can choose whether to add information to the deployment descriptor. If you choose to add information, you can provide the filter name and filter mappings for the filter. If you want to add initialization parameters for the filter, click the **`Next`** button. If you are ready to add the filter, click the **`Finish`** button.
7. If you click the **`Next`** button, in the next window you can add initialization parameters for the filter.
8. Click the **`Finish`** button to add the filter to the web application project.

The filter class opens in the NetBeans IDE source editor.

Deleting Filters

To delete a filter from the web application project, right-click the file node for the filter and select **`Delete`**. The IDE opens the Safe Delete dialog which has a **Safe Delete (with usage search)** check box. If you select the safe delete check box, the IDE presents the following additional buttons.

- **Preview**

  If you select this option the IDE shows all references to the filter along with the option to **Do Refactor** or **Cancel**.

- **Refactor**

  If you select this option the IDE deletes the filter and all references to the filter.

Web Application Frameworks

You can use frameworks within an existing web application project. You can enable frameworks when you first create a web application project in the NetBeans IDE. However, if you create an enterprise application project, you are never asked if you want the web application project to use a framework. You must add frameworks to the web application project.

Adding Frameworks

To add a framework to an existing web application, perform the following steps:

1. In the Projects window, right-click the web application project node and select **`Properties`**.
2. In the project properties window, select **`Frameworks`** from the Categories pane.
3. Click the **`Add`** button to add a framework to the web application project.
4. In the dialog box, select a framework (Spring Web MVC 2.5, JavaServer Faces, Struts 1.3.8, or Hibernate 3.2.5).
5. Click the **`OK`** button.

   If you add the JavaServer Faces (JSF) framework, you can determine how your project has access to JSF 2.0 libraries. Click the Configuration tab to specify how the Faces servlet will be registered in the project's deployment descriptor, and what the preferred project language will be (i.e., Facelets or JSP).

   If you add the Struts framework, you can specify the action URL pattern, application resource file, and specify whether to add the Struts TLDs to the web application.
6. Click the **`OK`** button in the project properties window to complete the addition of the framework.

The NetBeans IDE adds configuration files and libraries to the web application project as necessary.

Note – The IDE provides no utility for removing a framework from a web application project. This must be done manually.

Web Deployment Descriptors

The NetBeans IDE supports editing the standard web deployment descriptor.

Note – In Java EE 6, you typically do not need to use a deployment descriptor at all. However, you still need it for activities such as the configuration of welcome file lists, error pages, and security settings.

Creating the Standard Deployment Descriptor

By default, the New Web Application wizard does not create the `web.xml` file if your application will follow the Java EE 6 specification. Therefore, ifyou need this file anyway, you need to create it yourself.

To create the standard deployment descriptor (`web.xml` file) yourself, perform the following steps:

1. Right-click your web application project.
2. Select **`New`** > **`Other`** > Web > S**`tandard Deployment Descriptor`** menu option. Click Next.
3. Click the **`Finish`** button to create the `web.xml` file.
4. The deployment descriptor opens in the NetBeans IDE visual editor.

Opening the Standard Deployment Descriptor

To open the standard deployment descriptor, perform the following steps:

1. In the Projects window, expand your web application project.
2. Expand the Configuration Files node.
3. Double-click the `web.xml` file node.

The web deployment descriptor opens in the NetBeans IDE visual editor.

The visual editor groups the configuration settings for the `web.xml` file in the following tabs:

- General
- Servlets
- Filters
- Pages
- References
- Security
- XML

General Configuration

This tab of the visual editor organizes general information about the web application, context parameters, and web application listeners. This tab contains the following sections:

General Information

This section includes a display name, description, whether the web application is distributable, and a session timeout (in minutes).

Context Parameters

This section organizes the context parameters available in a web application. To create a context parameter, perform the following steps:

1. Click the **`Add`** button.
2. In the dialog window that appears, provide the following characteristics:
   - Parameter Name – The name for the parameter.
   - Parameter Value – The string value associated with the parameter name.
   - Description – An optional description for the parameter.
3. Click the **`OK`** button to create the context parameter.

To edit an existing context parameter, perform the following steps:

1. Select the parameter.
2. Click the **`Edit`** button.

3. In the window that opens, make the appropriate modifications.
4. Click the **OK** button to save the changes to the deployment descriptor.

To remove a context parameter, select the parameter and click the **Remove** button.

Note – When you click the **Remove** button, the IDE does not prompt you to confirm the deletion of the context parameter. If you accidentally remove a context parameter, use the IDE's undo functionality (**Edit > Undo**) to restore the deleted context parameter.

Web Application Listeners

This section organizes the web application listeners for the web application. To add a listener, perform the following steps:

1. Click the **Add** button.
2. In the dialog box that appears, supply the fully qualified class name for the listener (you can locate the class using the **Browse** button) and an optional description.
3. Click the **OK** button to add the listener to the deployment descriptor.

To edit an existing listener, perform the following steps:

1. Select the listener from the list.
2. Click the **Edit** button.
3. In the window that opens, modify the listener as necessary.
4. Click the **OK** button to save the change to the deployment descriptor.

To remove a listener, select the listener from the list and click the **Remove** button.

Note – When you click the **Remove** button, the IDE does not prompt you to confirm the deletion of the listener. If you accidentally remove a listener, use the IDE's undo functionality (**Edit > Undo**) to restore the deleted listener.

Servlet Configuration

This tab of the visual editor allows you to create or modify the servlet elements in the deployment descriptor. When you add a servlet in the IDE, you have the option of adding that servlet to the deployment descriptor. Any servlets that were added with this option appear as sections in this tab. To add an entry for a servlet, perform the following steps:

1. Click the **Add Servlet Element** button at the top of the window.
2. The IDE will open a dialog window. Figure 3-4 on page 3-24 illustrates the dialog box that is displayed. In this dialog box, supply the following characteristics:
   - Servlet Name – Indicates a logical name for the servlet.
   - Servlet Class – Use this field to specify the fully-qualified class name for the servlet. You can use the **Browse** button to locate the class in the project.
   - JSP File – Use this field to specify a JSP file instead of a servlet class. You can use the **Browse** button to locate the JSP file.
   - Description – Use this text area to supply an optional description for the servlet.
   - URL Pattern(s) – Provide the URL patterns (separated with commas) that map to the servlet.
3. Click the **OK** button to add the servlet to the deployment descriptor.

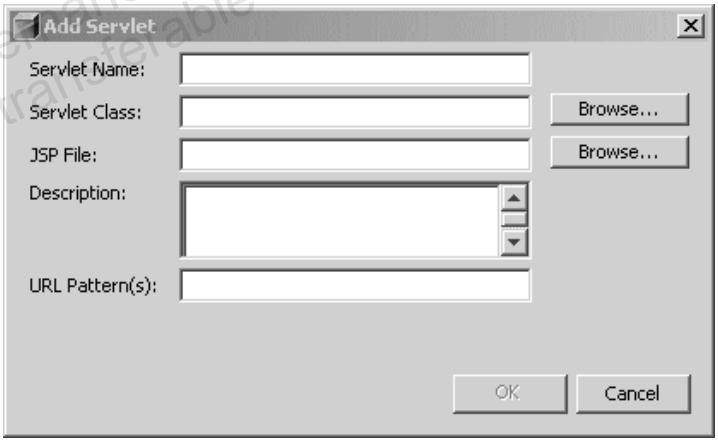


Figure 3-4 The Dialog to Add a Servlet to the Deployment Descriptor

The servlet entries appear as collapsible sections in the Servlets tab of the visual editor. You can expand and collapse these sections by clicking on the **+** or **-** symbols. For a servlet section, you can modify the servlet name, startup order, servlet class or JSP file, and URL patterns.

To remove a servlet element from the deployment descriptor, perform the following steps:

1. Click the **Remove** button in the header for that servlet element.
2. The IDE prompts you to confirm removal of the servlet entry and any related servlet mappings.
3. Click **OK** to complete the removal of the servlet entry from the deployment descriptor.

Note – When you remove a servlet entry from the deployment descriptor, the IDE does not delete the underlying servlet code. You need to delete the servlet from the Source Packages node yourself.

For each servlet, you can also specify initialization parameters. To add an initialization parameter, perform the following steps:

1. Click the **Add** button.
2. In the dialog box, provide the following characteristics:
   - Parameter Name – The name for the parameter.
   - Parameter Value – The string value associated with the parameter name.
   - Description – An optional description for the parameter.
3. Click the **OK** button to add the initialization parameter.

To edit an initialization parameter, perform the following steps:

1. Select the parameter in the list.
2. Click the **Edit** button.
3. In the window that opens, modify the initialization parameter as necessary.
4. Click the **OK** button to save the changes.

To remove an initialization parameter, select the parameter in the list and click the **Remove** button.

Note – When you click the **Remove** button, the IDE does not prompt you to confirm the deletion of the parameter. If you accidentally remove a parameter, use the IDE's undo functionality (**Edit > Undo**) to restore the deleted parameter.

For each servlet, you can also specify security role references used in the code. To add a security role reference, perform the following steps:

1. Click the **Add** button.
2. In the dialog box, provide the following characteristics:
   - Role Ref Name – Provide the role name used in the source code.
   - Role Ref Link – Select the security role defined in the deployment descriptor.
   - Description – Provide an optional description for the security role reference.
3. Click the **OK** button to add the security role reference.

To edit a security role reference, perform the following steps:

1. Select the role reference in the list.
2. Click the **Edit** button.
3. In the window that opens, modify the security role reference as necessary.
4. Click the **OK** button to save the changes.

To remove a security role reference, select the role reference in the list and click the **Remove** button.

Note – When you click the **Remove** button, the IDE does not prompt you to confirm the deletion of the security role reference. If you accidentally remove a security role reference, use the IDE's undo functionality (**Edit > Undo**) to restore the deleted security role reference.

Filter Configuration

The Filters tab of the visual editor allows you to add or modify the filters and filter mappings in the web application deployment descriptor. When you create a filter in the web application project using the IDE, you have the option of adding information to the deployment descriptor. Any filters that were added with this option appear as sections under the Servlet Filters section of this tab.

Servlet Filters

This section enables you to modify the filter name, description, and filter class for each servlet filter. Each filter entry appears as a collapsible section in the Servlet Filters section. You can expand and collapse these sections by clicking on the **+** or **-** symbols.

To add a filter entry to the deployment descriptor, perform the following steps:

1. Click the **Add Filter Element** button at the top of the window.
2. In the dialog box, supply the following characteristics:
   - Filter Name – Use this field to specify the logical name for the filter.
   - Filter Class – Use this field to specify the fully-qualified class name for the filter. You can use the **Browse** button to locate the class in the project.
   - Description – Use this text area to supply an optional description for the servlet.
3. Click the **OK** button to add the filter element.

To remove a filter element from the deployment descriptor, perform the following steps:

1. Click the **Remove** button in the header for that filter element.
2. You are asked to confirm removal of the filter entry and any associated filter mappings.
3. Click **OK** to complete the removal of the filter entry from the deployment descriptor.

For each filter, you can also specify initialization parameters. To add an initialization parameter, click the **Add** button. In the dialog, provide the parameter name, parameter value, and optional description. Click the **OK** button to add the initialization parameter. To edit an initialization parameter, select the parameter in the list and click the **Edit** button. Modify the initialization parameter as necessary, then click the **OK** button to save the changes. To remove an initialization parameter, select the parameter in the list and click the **Remove** button.

Filter Mappings

This section defines the filter mappings for the filters defined previously. To add a filter mapping, click the **Add** button. Figure 3-5 illustrates the add filter mapping dialog box. In this window, choose the filter name from the select list. Choose whether the filter applies to a particular URL pattern or a specific named servlet. Provide the URL pattern or choose the servlet. Finally, choose the dispatcher types for this filter mapping. When finished, click the **OK** button to create the filter mapping.

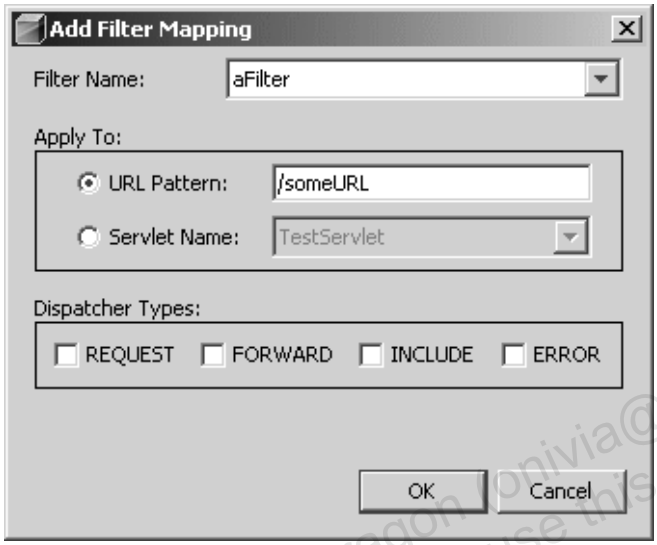


Figure 3-5 The Add Filter Mapping Dialog Box

To modify a filter mapping, select the mapping in the list and click the **Edit** button. Make your changes and click the **OK** button to save the changes. To remove a filter mapping, select the filter mapping in the list and click the **Remove** button.

Note – The visual editor does not allow you to specify the order of the filter mappings. To modify the order, you must edit the raw XML in the deployment descriptor.

Page Configuration

The Pages tab of the visual editor provides the ability to configure the welcome pages, error pages, and JSP property groups for the web application.

Welcome Files

This section allows you to specify a comma-delimited list of welcome files for the web application. You can use the **Browse** button to locate and specify the welcome files.

Error Pages

This section lists the custom error pages for the web application. To add an error page, click the **Add** button. In the dialog, click the **Browse** button to select the error page, provide either the error code or exception type, and click the **OK** button to add the error page to the deployment descriptor. To edit an existing error page entry, select the error page in the list and click the **Edit** button. Modify the error page entry accordingly and click the **OK** button to save your changes. To delete an error page entry, select the entry from the list and click the **Remove** button.

JSP Property Groups

This section allows you to configure JSP property groups for the web application. Click the **Add JSP Property Group** button to add a property group. In the dialog box, specify the display name, optional description, and a comma-delimited list of URL patterns. Click the **OK** button to create the JSP property group entry.

The JSP property group entries appear as collapsible sections in the JSP Property Groups section. You can expand and collapse these sections by clicking on the **+** or **-** symbol. For a JSP property group section, you can modify the display name, description, URL patterns, page encoding, whether to ignore expression language, whether to ignore Java scripting elements, whether to require XML syntax, the preludes and codas for the pages in that group.

To delete a JSP property group entry, click the **Remove** button in the header for that JSP property group. You are prompted to confirm the deletion of the JSP property group. Click **OK** to complete the removal of the JSP property group from the deployment descriptor

Reference Configuration

The References tab of the visual editor provides the ability to configure references that are available within the web application. These references include environment entries, resource references, resource environment references, EJB references, and message destination resources. Each of these are represented as collapsible sections on the screen.

Environment Entries

This section provides access to variables in the web application's JNDI namespace. Click the **Add** button to add a new environment entry to the deployment descriptor. In the dialog, provide the entry name, entry data type, entry value, and optional description. Click the **OK** button to complete the addition of the entry. To edit an environment entry, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete an environment entry from the deployment descriptor, select the entry in the list and click the **Remove** button.

Resource References

This section provides access to resources in the web application's JNDI namespace. Click the **Add** button to add a new resource reference entry to the deployment descriptor. In the dialog box, provide the resource name, resource data type, authentication requirements, sharing scope, and optional description. Click the **OK** button to complete the addition of the entry. To edit a resource reference entry, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete a resource reference entry from the deployment descriptor, select the entry in the list and click the **Remove** button.

Resource Environment References

This section provides access to administered objects in the web application's JNDI namespace. Click the **Add** button to add a new resource environment reference entry to the deployment descriptor. In the dialog box, provide the resource name, resource type (`javax.jms.Queue` or `javax.jms.Topic`), and optional description. Click the **OK** button to complete the addition of the entry. To edit a resource environment reference entry, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete a resource environment reference entry from the deployment descriptor, select the entry in the list and click the **Remove** button.

EJB References

This section provides access to EJB components in the web application's JNDI namespace. Click the **Add** button to add a new EJB reference entry to the deployment descriptor. In the dialog box, provide the EJB reference name, EJB type (`Session` or `Entity`), interface type (Local or Remote), fully qualified home interface class (not required for EJB 3.0 session beans), fully qualified remote (or local) interface class, linked EJB name, and optional description. Click the **OK** button to complete the addition of the entry. To edit an EJB reference entry, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete an EJB reference entry from the deployment descriptor, select the entry in the list and click the **Remove** button.

Message Destination References

This section provides access to message destination resources in the web application's JNDI namespace. Click the **Add** button to add a new message destination reference entry to the deployment descriptor. In the dialog box, provide the reference name, reference type (`javax.jms.Queue` or `javax.jms.Topic`), usage (consumes, produces or both), link to the message destination, and optional description. Click the **OK** button to complete the addition of the entry. To edit a message destination reference entry, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete a message destination reference entry from the deployment descriptor, select the entry in the list and click the **Remove** button.

Security Configuration

The Security tab of the visual editor allows you to configure security settings for login configurations, security roles, and security constraints for the web application. Each of these are represented as collapsible sections on the screen.

Login Configuration

This section enables you to configure the security login configuration for the web application. You can expand and collapse the Login Configuration section by clicking on the **+** or **-** symbol.

Select one of the following login types.

- None – Select this if you do not want to use authentication. This is the equivalent of removing the login configuration element from the deployment descriptor.
- Digest – Select this to use an authentication mechanism in which a Web application authenticates itself to a Web server by sending the server a message digest along with its HTTP request message.
- Client Certificate – Select this to use an authentication mechanism that uses HTTP over SSL. The server and, optionally, the client authenticate each other with a public key certificate.
- Basic – Select this to use the default browser login. To use this type, you need to specify the name of the security realm. Enter the name of the security realm in the Realm Name field. The security realm must also be defined on the application server. There is no validation for this value.
- Form – Select this to use a form page to login. You need to specify the JSP pages for the login and error pages.

Security Roles

This section enables you to configure the security roles for the web application. Click the **Add** button to add a new security role to the deployment descriptor. In the dialog, provide the name of the security role and optional description. Click the **OK** button to complete the addition of the entry. To edit a security role, select the entry in the list and click the **Edit** button. Make the necessary changes and click the **OK** button to save those changes. To delete a security role from the deployment descriptor, select the entry in the list and click the **Remove** button.

Security Constraints

This section enables you to configure the security constraints for the web application. The security constraint is used to specify the secure areas that each security role can access. Click the **Add Security Constraint** button to add a security constraint. Each security constraint appears as a collapsible section in the Security Constraint section. You can expand and collapse these sections by clicking on the **+** or **-** symbol.

To configure a security constraint, perform the following steps:

1. Enter the name of the security constraint.
2. Click the **Add** button to add a Web Resource Collection.

3. In the dialog box, supply the following characteristics:
   - Resource Name – Use this field to specify the name for the web resource.
   - Description – Use this text area to supply an optional description for the web resource.
   - URL Pattern(s) – Use this field to specify the URL pattern specifying the secure area for the security constraint.
   - HTTP Method(s) - Select the HTTP method used for the resource.
4. Click the **OK** button in the dialog box to add the web resource.
5. Select Enable Authentication Constraint to define role names that can access the security constraint. Click the **Edit** button to choose from the defined security roles.
6. Select Enable User Data Restraint to define how data between a web client and the web container should be protected. Select a transport guarantee type and supply an optional description.

To edit a security constraint, expand the security constraint section and make the necessary changes to the security constraint properties. To delete a security constraint from the deployment descriptor, select the security constraint entry in the list and click the **Remove** button.

XML Editor

The XML tab of the visual editor allows you to view and edit the raw XML elements in the deployment descriptor.

GlassFish Server Web Deployment Descriptor

The following sections describe how to open and configure the GlassFish Server web deployment descriptor.

Opening the GlassFish Server Web Deployment Descriptor

To configure the GlassFish Server web deployment descriptor, double-click the `sun-web.xml` file node under the Configuration Files node of the web application project. The Sun-specific deployment descriptor opens in the NetBeans IDE visual editor.

To open the Sun-specific deployment descriptor in the NetBeans IDE XML editor from the Projects window, right-click the `sun-web.xml` file node and select **`Edit`**. The **`Edit`** contextual menu item is not available if the `sun-web.xml` file is already open in the visual editor.

Note – If the `sun-web.xml` file is already open in the visual editor, you can open the file in the NetBeans IDE XML editor by clicking the **`XML`** button at the top of the visual editor.

Configuring the GlassFish Server Web Deployment Descriptor

In the left pane of the visual editor, you can select the **`Sun Web Application`** node or **`Web Services`** node to configure general information about the web application's behavior on the GlassFish Server.

When the **`Sun Web Application`** node is selected, the visual editor groups the configuration settings for the `sun-web.xml` file in the following tabs located at the bottom of the visual editor:

- General - In the General tab you can configure the web module and class loader.

  For the module, you can set the context root, specify an error URL and HTTP servlet security provider, and JSP configuration properties for the web application.

  For the class loader, you can enable a class loader configuration, set an additional classpath, and other class loader properties for the web application.
- Servlets - In the Servlets tab you can add principal names associated with run-as fields and web service end points for each servlet in the web application.
- Security - In the Security tab you can assign principals and groups to security roles.
- Web Services - In the Web Services tab you can configure web services and web service clients.
- Messaging - The Message tab helps you provide a reference to any resource from which a web application is going to send or receive messages.
- Environment - In the Environment tab, you can set EJB references, resource references, resource environment references, and message destination references.

JavaServer Faces

Web Applications with JSF Framework Support

You can create new web applications with JSF support or add JSF support to existing web applications.

Creating Web Applications with the JSF Framework

To add a framework to an existing web application, perform the following steps:

1. Choose File > New Project from the IDE's main menu. In the New Project wizard, select the Java Web category. Under Projects, select Web Application and then click Next.
2. Type in a name for the project in the Project Name field, specify the location of the project in the Project Location field, and click Next.
3. Select the server you want to use with the project from the Servers drop-down list. (Servers in the drop-down list are registered with the IDE. To register a new server, click the Add button.)
4. In the Java EE Version drop-down, select the Java EE version you want to apply to the project. Specify the context path for the application in the Context Path text field. Click Next.
5. In the Frameworks panel, select JavaServer Faces. When you make the selection, various configuration options become available to you. You can determine how your project has access to JSF 2.0 libraries. Click the Configuration tab to specify how the Faces servlet will be registered in the project's deployment descriptor.
6. Click Finish. The IDE generates the new project and adds JSF-specific configuration files and libraries to the project as necessary.

Adding the JSF Framework to an Existing Project

See the section Web Application Frameworks > Adding Frameworks.

Working with JSF Pages

Creating JSF Pages

Use the JSF Page wizard to create Facelets and JSP pages for your project. In JSF 2.0, Facelets is the preferred way to declare JSF pages. The Facelets option in the wizard is selected by default, however you can still use the wizard to create JSP pages (.jsp files) and JSP fragments (.jspf files).

1. Choose File > New Project from the IDE's main menu. In the New Project wizard, select the Java Web category. Under Projects, select Web Application and then click Next.
2. In the Name and Location step of the wizard, type in a name for the file, and specify a location for the file within your project. In the read-only Created File field, you see the full path to the file that is about to be created by the wizard.
3. Under Options, select either the Facelets (default) or the JSP File option. If you select the JSP File option, you can further specify whether to create a JSP fragment by ticking the Create as a JSP Segment checkbox. Click Finish.

Editing JSF Pages

You can edit JSF pages using the IDE's source editor. From the Projects window, you can double-click any JSF page node to open it in the editor. The IDE's editor provides the following support when editing Facelets pages.

Code Completion:

- JSF and Facelets tags: Press Ctrl-Space on JSF and Facelets tags to invoke a pop-up listing valid tag entries for completing a tag you've begun typing.
- JSF namespaces: If you apply code-completion for a tag whose namespace is not yet declared in the file, the namespace is automatically added to the page's <html> tag.
- JSF namespace paths: If you manually declare a namespace, press Ctrl-Space between the quotes of a namespace declaration to invoke a list of possible paths.

- EL (Expression Language) expressions: When typing EL expressions in the editor, press Ctrl-Space to call up a list of possible entries for completing the expression. The pop-up list typically includes JSF managed beans and their properties, property bundle messages, and implicit objects.

Documentation: Press Ctrl-Space on JSF and Facelets tags to invoke a documentation pop-up that describes the given tag.

Hints and Error Messages: The editor provides you with warning and error messages while you edit JSF pages. Errors display with a red badge in the left margin of the editor, and the corresponding code is underlined in red. You can hover your mouse over the badge or underlined code to view a tooltip describing the cause of the error. The editor checks whether:

- a declared library exists
- the library matched by the tag prefix contains such a component or tag
- the tag contains all required attributes
- all entered attributes are defined in the component's interface
- undeclared components exist on the page
- taglib declarations exist without usages

Syntax Highlighting:

- JSF and Facelets tags display in blue.
- Tag attributes display in green.
- Tag attribute values display in orange.
- EL expressions display in black, with light-green background.

Deleting JSF Pages

Take the following steps to delete a JSF page.

1. Right-click the page node in the Projects window and choose Delete. The IDE presents you with a Confirm Object Deletion dialog to confirm whether you want to delete the file.
2. Click Yes. The node is removed from the project, and if the page was previously opened in the editor, it disappears.

Designing JSF Pages

You can make use of JSF form and table components listed in the Palette (Window > Palette). Components in the Palette represent code snippets which you can drag-and-drop into the opened file in the Source Editor. The JSF components included in the Palette can equally be accessed by typing 'jsf' in the editor, then pressing Ctrl-Space.

Adding JSF Forms

You can add a JSF form to a Facelets or JSP page by:

- dragging the JSF Form component from the Palette (Window > Palette) onto the page
- typing 'jsf' in the editor, pressing Ctrl-Space, then choosing the JSF Form option.

The following code snippet is added to the page:

```
<f:view>
   <h:form>
   </h:form>
</f:view>
```

The JSF Form component requires that you add the JSF core and html tag libraries to your page's root element (typically the <html> tag):

```
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
```

Creating Facelets Templates

Use the Facelets Template wizard to create and add a Facelets template to your project. The wizard creates an XHTML template file using <h:head> and <h:body> tags, and places associated stylesheets in the resources/css folder of your application's web root. You can choose from eight unique layout styles, and specify whether the layout is implemented using CSS or an HTML <table> element.

To create a new Facelets template:

1. In the Projects window, right-click your project node and choose New > Facelets Template. The Facelets Template wizard opens. (If Facelets Template is not listed, choose Other. Then select the JavaServer Faces category and Facelets Template file type. Click Next.)
2. In File Name, type in a name for the template.
3. In Location, specify the top-level location for the file. ('Web Pages' is the default option, and places the file in the given project's web root.)
4. In Folder, specify a folder within the selected location, if you require. The Created File field provides a read-only path to the new location for the template file.
5. For Layout Style, specify whether you want the layout to be generated using CSS styles or an HTML table.
6. Select the icon that corresponds to the layout you want generated. (A black layout icon indicates that the icon is selected.)

Click Finish. The new Facelets template is generated and opens in the editor. The wizard generates a `default.css` file, and a `cssLayout.css` or `tableLayout.css` file, depending on your layout selection.

Calling Facelets Templates

To call a Facelets template from another Facelets page, you need to reference the template using the <ui:composition> tag's template attribute, and define any areas that are declared in the template using the <ui:define> tag.

The IDE provides a Facelets Template Client wizard that generates a new Facelets page containing code necessary to call a specified Facelets template.

To create a new Facelets page that calls a Facelets template:

1. In the Projects window, right-click your project node and choose New > Facelets Template Client. The Facelets Template Client wizard opens. (If Facelets Template Client is not listed, choose Other. Then select the JavaServer Faces category and Facelets Template Client file type. Click Next.)
2. In File Name, type in a name for the Facelets page.

3. In Folder, specify a folder within the project, if you require. The default location is your project's the web root (This is the project's web folder.) The Created File field provides a read-only path to the new location for the file.
4. In Template, enter the path to the Facelets template that you want this page to refer to. Click the Browse button to navigate to the template file in a dialog box.
5. Select the generated root tag. (<html> is the default option).

Click Finish. The new Facelets template client is generated and opens in the editor. Within the new template client, you can specify the content you want to display for each template area, defined by the <ui:define> tags.

Managed Beans

Creating Managed Beans

You can create JSF managed beans for your application using the IDE's Managed Bean wizard. In JSF 2.0, any metadata that you specify in the wizard is translated into annotations that are applied to the managed bean once it is generated. If you prefer to register the managed bean with the application using a Faces configuration file (faces-config.xml) instead, the wizard provides the option of doing so.

To create a new JSF managed bean:

1. In the Projects window, right-click your project node and choose New > JSF Managed Bean. The JSF Managed Bean wizard opens. (If JSF Managed Bean is not listed, choose Other. Then select the JavaServer Faces category and JSF Managed Bean file type. Click Next.)
2. In Class Name, type in a name for the managed bean.
3. In Location, specify the top-level location for the file. ('Source Packages' is the default option, and places the file in the project's src/java folder.)
4. In Package, specify a package name for the new class. If you specify the name of a package that does not exist, it will be created upon completing the wizard. The Created File field provides a read-only path to the new location for the file.

5. (Optional.) If you want to to register the managed bean with the application using a Faces configuration file, select the 'Add data to configuration file' option. Doing so will prevent the wizard from generating annotations in the new class. If a faces-config.xml does not yet exist for the application, one will be created upon completing the wizard.
6. (Optional.) Specify configuration details:
   - Name: Specifies the name which the managed bean can be invoked by. If left blank, the managed bean can be accessed using the name of the managed bean class, but with the first letter in lowercase.
   - Scope: Specifies the scope into which a newly created instance of the specified managed bean will be stored. (Default is request; choose none if you want to refrain from placing bean instances in any scope).
   - Description: A textual description of the element, which can be flagged with a language code using the xml:lang attribute.
7. Click Finish. The new managed bean is generated and opens in the editor.

JSF Configuration

Adding JSF Configuration

JSF applications can be configured using a Faces configuration file (`faces-config.xml`). In JSF 2.0 applications, some configuration can be accomplished using annotations. You can create a Faces configuration file for your project using the Faces Configuration wizard.

To create a Faces configuration file:

1. Right-click the project node in the Projects window and choose New > JSF Faces Configuration. (If this option is not listed, choose Other. In the File wizard that displays, choose the JavaServer Faces category, then JSF Faces Configuration. Click Next.)
2. Enter a name for the file. (`faces-config` is the default name.)
3. Click Finish. A new Faces configuration file is generated in the project's `WEB-INF` directory.

Configuring Managed Beans upon Creation

The Managed Bean wizard configures managed beans when you create them. Depending on the JSF version you are using and whether your project contains a `faces-config.xml` file, the following rules apply:

- **JSF 2.0 applications.** Your entries in the wizard are automatically converted into annotations that are included in the bean class.
- **JSF 2.0 applications containing a `faces-config.xml` file.** The wizard provides you with the option of choosing whether to configure using annotations or the `faces-config.xml` file.
- **JSF 1.2 applications.** Your project must contain a `faces-config.xml` file in order to successfully use the Managed Bean wizard.

Registering Existing Java Classes as Managed Beans

If you are using a Faces configuration file, you can use the Add Managed Bean dialog box to register an existing Java class as a managed bean. The Add Managed Bean dialog box is accessible from the right-click menu of the configuration file.

To configure an existing Java class as a managed bean:

1. Open the project's `faces-config.xml` file in the editor.
2. Right-click in the `faces-config.xml` file and choose Insert > Managed Bean.
3. In the dialog, enter the following details:
    - Bean Name: Specifies the name which the managed bean can be invoked by.
    - Bean Class: Specifies the fully qualified class name of the Java class that will be used to instantiate a new instance, if creation of the specified managed bean is requested. The specified class must conform to standard JavaBean conventions. In particular, it must have a public, no-argument constructor, and properties must be accessible using get and set, following standard naming conventions.
    - Scope: Specifies the scope into which a newly created instance of the specified managed bean will be stored (default is request; choose none if you want to refrain from placing bean instances in any scope).

- Bean Description: An optional textual description of the element.

4. Click Add. Based on the details in the dialog, code is generated and added to the Faces configuration file.

Adding Navigation Rules

If you are using a Faces configuration file, you can use the Add Navigation Rule and Add Navigation Case dialog boxes to add JSF navigation rules to your application. Both dialog boxes are accessible from the right-click menu of the configuration file.

Navigation Rules

A JSF *navigation rule* can contain numerous *navigation cases*. A navigation rule is bound to the view from which a request originates.

To add a navigation rule to the Faces configuration file:

1. Open the project's `faces-config.xml` file in the editor.
2. Right-click in the `faces-config.xml` file and choose Insert > Navigation Rule.
3. In the dialog, enter the following details:
   - Rule from View: (`<from-view-id>`) Specifies the identifier of the view from which the request originated.
   - Rule Description: (`<description>`) An optional textual description of the element.
4. Click Add. Based on the details in the dialog, code is generated and added to the Faces configuration file.

Navigation Cases

A navigation case applies a specific logical outcome to a destination view, and is bound to a navigation rule.

To add a navigation case to an existing navigation rule:

1. Open the project's `faces-config.xml` file in the editor.
2. Right-click within the navigation rule you want to apply the case to and choose Insert > Navigation Case.
3. In the dialog, enter the following details:

- **From View:** (`<from-view-id>`) Specifies the identifier of the view for which the containing navigation rule is relevant. If no value is specified, the rule applies to navigation decisions for all views.
- **From Action:** (`<from-action>`) Specifies an action reference expression that must have been executed (by the default `ActionListener` for handling application level events) in order to select this navigation rule. If not specified, this rule will be relevant no matter which action reference was executed (or if no action reference was executed).
- **From Outcome:** (`<from-outcome>`) Specifies an outcome string returned by the execution of an application action method selected via an `actionRef` property (or a literal value specified by an `action` property) of a `UICommand` component. If specified, this rule is relevant only if the outcome value matches the element's value. If not specified, the rule is relevant no matter what the outcome value was.
- **To View:** (`<to-view-id>`) Specifies the identifier of the view that should be displayed if the navigation rule is matched.
- **Redirect:** (`<redirect>`) Specifies that navigation to the view id should be accomplished by performing an HTTP redirect rather than the usual `ViewHandler` mechanisms.
- **Rule Description:** (`<description>`) Specifies a textual description of the element it is nested in.

4. Click Add. Based on the details in the dialog, code is generated and added in the Faces configuration file to the specified navigation rule.

Java Persistence API

The NetBeans IDE provides support for Java Persistence API (JPA) entity classes. JPA can be used in Java SE applications, Web modules, and EJB modules.

Creating Persistence Units

To create a persistence unit, perform the following steps:

1. Right-click the EJB module project and select **`New > Other`**.
2. Choose **`Persistence`** from the Categories pane and **`Persistence Unit`** from the File Types pane.
3. Click the **`Next`** button.
4. Provide the following information on the next screen:
   - Persistence Unit Name – The logical name for the persistence unit.
   - Persistence Provider - The persistence provider to be used. You can select New Persistence Library from the drop-down list to configure a different provider.
   - Data Source – The JNDI name of the JDBC resource that is used for persistence. You can select New Data Source from the drop-down list to configure a new JDBC resource.
   - Use Java Transaction APIs – If checked, the container use the Java Transaction API to manage the transactions involving the entities.
   - Table Generation Strategy – This is used to indicate if database tables should be created when the application is deployed. You have three choices:
     - Create – The database tables are created if they do not exist.
     - Drop and Create – Existing database tables are dropped (and any data they contain is lost) and then the database tables are created.
     - None – Database tables are not created. The tables must already exist in the database.
5. Click the **`Finish`** button to create the persistence unit.
6. The IDE creates the `persistence.xml` file in the project. You can find this file under the Configuration Files node of the project.

Creating Entity Classes

To create an entity class, perform the following steps:

1. Right-click the EJB module project and select **New > Other**.
2. Choose **Persistence** from the Categories pane and **Entity Class** from the File Types pane.
3. Click the **Next** button to continue.
4. In the next window of the wizard, provide the following information:
   - Class Name – Provide the Java class name for the entity class.
   - Location – Choose the location for the source files (if multiple source directories are configured).
   - Package – Provide a package name for the entity class.
   - Primary Key Type – Specify the primary key for the entity class. You can click the ellipsis (...) button to locate a class.
5. If you have not created a persistence unit in the module, you can create the persistence unit by clicking the **Create Persistence Unit** button. This button is not present if the module already has a persistence unit.
6. Click the **Finish** button to create the entity class in the EJB module.

The IDE creates the entity class and opens the entity class in the NetBeans IDE source editor.

Creating Entity Classes From a Database

To create entity classes from a database, perform the following steps:

1. Right-click the EJB module project and select **New > Other**.
2. Choose **Persistence** from the Categories pane and **Entity Classes from Database** from the File Types pane.
3. Click the **Next** button to continue.
4. In the next window of the wizard, select one of the following to display the list of database tables for which you can generate entity classes.

- Data Source – Select this if you want to generate entity classes from a connected data source. Select a data source from the drop-down list of data sources registered with the IDE. If your data source is not listed, you can choose **`New Data Source`** from the list to create one.
- Database Schema – Select this if you want to generate entity classes from a database schema. This option is not available if your project does not have a database schema.

5. From the list of available tables in the right pane, select the tables you want represented by entity classes and click **`Add`**. When you click **`Add`**, the selected tables and any referenced tables are listed in the left pane of the wizard.
6. Click the **`Next`** button to continue.
7. In the next window of the wizard, confirm the names of the entity classes to be generated and specify the location and package where you want the entity classes created.
8. Click the **`Finish`** button to create the entity class in the EJB module.

The IDE creates the entity classes in the specified location.

EJB Modules

The NetBeans IDE provides full support for EJB 3.1 components. These can be created within web applications or as EJB modules within Java EE applications.

Creating EJB Modules in a Java EE Application

To create a new EJB module in a Java EE application, perform the following steps:

1. Select New Project from the File menu.
2. In the wizard, select **`Enterprise`** from the Categories pane and **`EJB Module`** from the Projects pane.
3. Click the **`Next`** button.
4. On the next screen, provide the following information:
   - Project Name – The project name is used as the name for the project directory. For cross-platform compatibility, this name should not contain spaces.
   - Project Location – The location is the base directory for the project. The project folder is based on the project name. The full path is displayed in the Project Folder field.
   - Add to Enterprise Application – You can choose to add this EJB module to an existing enterprise application.
   - Server – Select the application server that is used for deployment of this application.
   - Java EE Version – Choose the appropriate Java EE version for the EJB module (J2EE 1.4 or Java EE 5).
   - Main Project – If checked, this project is marked as the main project within the IDE.
5. Click the **`Finish`** button to create the EJB module.

You can also create an EJB module as part of an enterprise application. In this case, the name of the EJB module is specified in the wizard used to create the enterprise application. The EJB module project will be a subdirectory of the enterprise application project.

Session Beans

The NetBeans IDE provides support for stateless, stateful, singleton, and no-interface view session beans. These can be created either within web applications or within EJB modules.

Creating Session Beans

To create a new session bean in the IDE, perform the following steps:

1. in the Projects window, right-click the project node of the application where you want to create your session bean and select **New > Other**.
2. Select **Java EE** from the Categories pane and **Session Bean** from the File Types pane.
3. Click the **Next** button to continue.
4. In the next window of the wizard, provide the following information:
   - EJB Name – Type the logical name for the EJB component. This is the base name for the class and business interface.
   - Location – Choose the location for the source files (if multiple source directories are configured).
   - Package – Provide the package name for the session bean.
   - Session Type – Choose either Stateless, Stateful, or Singleton. The choice determines the annotation created in the bean class.
   - Create Interface – Choose Remote, Local or both. If you choose Local, a business interface named *EJBName*`Local` is created in the package. If you choose Remote, a business interface named *EJBName*`Remote` is created in the package.
5. Click the **Finish** button to create the session bean in the EJB module.

The IDE creates the files that compose the session bean and creates a `sun-ejb-jar.xml` deployment descriptor file, if necessary. The new bean class opens in the NetBeans IDE source editor.

Adding Business Methods

To add a method to a session bean, perform the following steps:

1. Right-click in the source editor for your bean class and select **EJB Methods > Add Business Method**.

2. The Add Business Method dialog window is displayed. Figure 3-6 on page 3-51 illustrates this window.
3. Provide the following information in the dialog window:
   - Name – The name for the business method.
   - Return Type – The return type for the business method. You can select the return type from the drop-down list, click the browse button to look for a Java class, or simply type the class name in the field provided.
   - Use in interface – The business interface(s) in which the business method will be listed. You can select Local, Remote, or both (if the bean was created with both interfaces).
   - Input Parameters and Exceptions – The Parameters tab lets you specify the parameters and their order in the business method. Use the **Add** button to add parameters. The Exceptions tab lets you specify the exceptions thrown by the business method. While on the Exceptions tab, use the **Add** button to add exceptions to the business method.
4. Click the **OK** button to add the business method.

The IDE adds the business method to the appropriate business interface and the bean class.

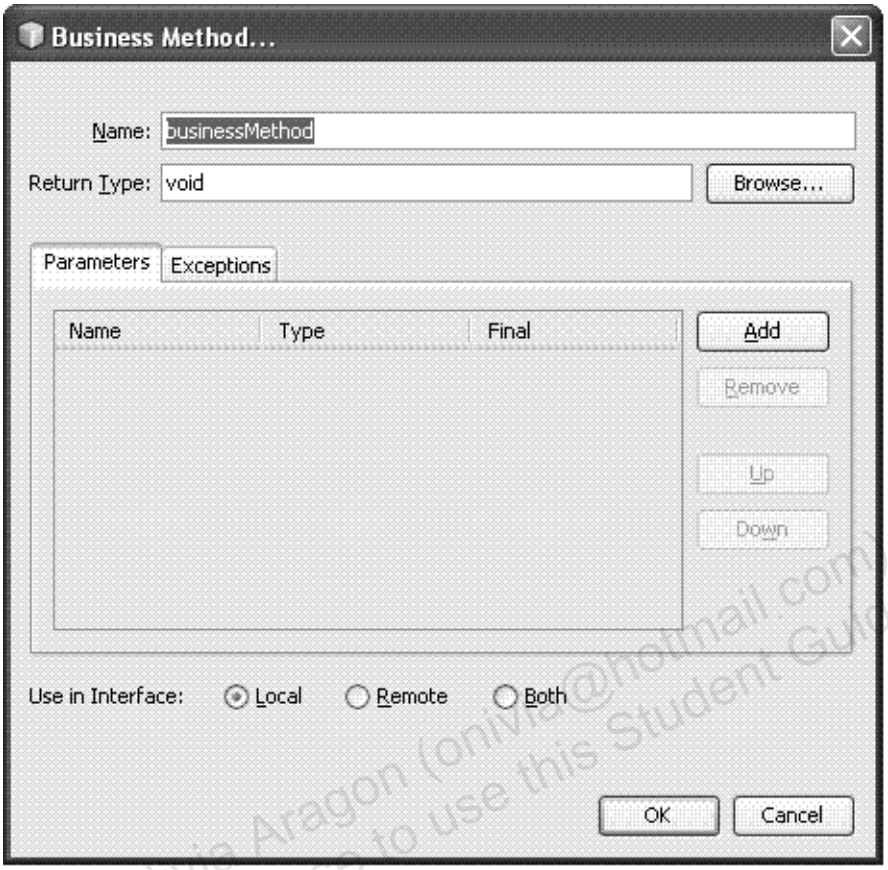


Figure 3-6 The Dialog Window for Adding a Business Method

Removing Methods

To remove a method from an EJB component, you must manually delete the method from the bean class and interfaces using the IDE source code editor.

Message-Driven Beans

To create a new message-driven bean in the IDE, perform the following steps:

1. Right-click the EJB module project node in the Projects window and select **`New > Other`**.
2. Select **`Enterprise`** from the Categories pane and **`Message-Driven Bean`** from the File Types pane.

3. Click the **Next** button to continue.
4. In the next window of the wizard, provide the following information:
   - EJB Name – Type the logical name for the message-driven bean.
   - Location – Choose the location for the source files (if multiple source directories are configured).
   - Package – Provide a package name for the bean class.
   - Project Destinations check box – Select

     Select a destination from the drop-down list of destinations registered with the IDE. If your destination is not listed, you can choose **Add** to create one. To add a destination you must provide the destination name and type (queue or topic).
5. Click the **Finish** button to create the message-driven bean in the EJB module.

The IDE creates the bean class opens it in the NetBeans IDE source editor.

When you create a message-driven bean, the IDE also automatically creates the necessary JMS resources for the bean in the EJB module project. The name of the created JMS resource is based on the name of the message-driven bean.

The JMS resources can be found in the Projects window under the Server Resources node of the EJB module project.

The IDE registers the JMS resources with the GlassFish Server when the project is deployed.

Configuring EJB Deployment Descriptors

The NetBeans IDE supports editing the standard EJB deployment descriptor. If necessary, you can use the IDE to create and modify the deployment descriptor for an EJB module project.

Creating the Standard EJB Deployment Descriptor

To create a standard EJB deployment descriptor, perform the following steps:

1. Right-click the EJB module project and select **New > Other**.

2. Choose **`Enterprise`** from the Categories pane and **`Standard Deployment Descriptor`** from the File Types pane.
3. Click the **`Next`** button.
4. On the next screen, select the location where you want the IDE to create the deployment descriptor for the EJB module.
5. Click the **`Finish`** button to create the deployment descriptor in the EJB module.

Opening the Standard EJB Deployment Descriptor

To open the standard EJB deployment descriptor, perform the following steps:

1. In the Projects window, expand your EJB module project.
2. Expand the Configuration Files node.
3. Double-click the `ejb-jar.xml` file node.

The EJB deployment descriptor opens in the NetBeans IDE XML editor.

The XML Editor

The XML editor is used to examine or manually edit the XML elements in the deployment descriptor. The raw XML of the deployment descriptor is displayed in the XML editor when you open the deployment descriptor. You must use this editing mode for some settings, such as security and transaction information.

The GlassFish Server EJB Deployment Descriptor

To open the GlassFish Server EJB deployment descriptor, expand the Configuration Files node in your EJB module project and double-click the `sun-ejb-jar.xml` file node. In the visual editor, you can configure module settings and EJB component-specific settings.

- Module Settings – Select the General tab in the visual deployment descriptor editor. This is where you provide the optional name for the EJB module.
- EJB Settings – Select the EJB tab in the visual deployment descriptor editor. Use the ensuing pane to specify the JNDI name and additional configuration information for the EJB component.

- Security Settings – Select the Security tab in the visual deployment descriptor editor. Use the ensuing pane to specify security role mappings for the EJB component.
- EJB Settings – Select the Web Services tab in the visual deployment descriptor editor. Use the ensuing pane to expose web services.
- EJB Settings – Select the Web Services tab in the visual deployment descriptor editor. Use the ensuing pane to add message destinations.

If the `sun-ejb-jar.xml` deployment descriptor is already open in the visual editor and you need to edit the raw XML, click the **`Edit As XML`** button at the top of the visual editor.

To open the Sun-specific EJB deployment descriptor in the NetBeans IDE XML editor from the Projects window, right-click the `sun-ejb-jar.xml` file node and select **`Edit`** from the contextual menu.

Web Services

The NetBeans IDE supports the development of RESTful and SOAP web services, according to the JAX-RS and JAX-WS specifications. It supports web service creation in both Java EE 6 and Java EE5 web applications. In addition to JAX-RS and JAX-WS, the IDE supports the older J2EE 1.4 JAX-RPC standard through an optional plugin.

In addition, the IDE also supports the generation of a test application for trying out your web services, as well as full-blown client-side code for consuming web services in Java SE or web applications.

For JAX-WS web services, the IDE enables you to choose to create a web service according to your approach to developing the web service: Java-to-web services description language (WSDL) or WSDL-to-Java technology. If you do not have a WSDL file on which to base your web service, the IDE can help you create an empty web service or a web service based on a session bean. If you have a WSDL file, the IDE can help you generate the web service based on the WSDL file.

JAX-RS Web Services

NetBeans IDE has wizards for creating RESTful web services and their clients. These services are compliant with the latest JSR 311 (now JAX-RS) specification.

Creating a RESTful Web Service

You can create RESTful web services either from a pattern or from entity classes. If you want to create a RESTful web service from a pattern, you have the following patterns to choose from:

- **Simple Root Resource.** A RESTful root resource class with GET and PUT methods. Useful for creating wrapper services for invoking SOAP (WSDL-based) web services. Note that in previous versions of NetBeans IDE, this pattern was called "singleton."
- **Container-Item.** A pair of RESTful resource classes consisting of an item resource class and its container resource class. Item resources can be created and added to the container resource using the POST method on the container resource class. Note that the URI for the newly created item resource is determined by the container resource.

- **Client-Controlled Container-Item.** A pair of RESTful resource classes consisting of an item resource class and its container resource class. This pattern is a slight variation of the Container-Item pattern. The difference is that there is no POST method on the container resource class for creating item resources. Instead, item resources are created using the PUT method on the item resource class. The reason this is called the Client-Controlled Container-Item pattern is because the URI for the item resource is determined by the client and not the container resource.

With NetBeans IDE, you create RESTful web services inside Java Web applications. You have a choice of three wizards, launched from New > Web Services: RESTful Web Service from Patterns, RESTful Web Service from Database, and RESTful Web Service from Entity Classes. The RESTful Web Service from Database wizard combines the RESTful Web Service from Entity Classes wizard with the Entity Classes from Database wizard, so you can create entity classes and web services in one procedure.

Creating a RESTful Web Service from a Database

The following procedure shows how to create entity classes and a RESTful web service from a database. This is the most common use case.

1. Create a Java Web application. Name it CustomerDB. You can create either a Java EE 5 or a Java EE 6 application, but for illustration purposes, create CustomerDB as a Java EE 6 application. Select the GlassFish v3 application server.
2. In the Projects window, right-click the CustomerDB project node and select New > Web Services > RESTful Web Service From Database.
3. When the wizard opens, select the jdbc/sample data source. The JavaDB server, bundled with GlassFish v3, starts.
4. After the JavaDB server has started, add the Customer table to the entity classes. The Discount_Code table is then added automatically.
5. Click Next. The Entity Classes page opens. Accept the default file and folder names for the entity classes. Name the package customerdb and create a persistence unit. Create the default persistence unit that you are offered when you open the Create Persistence Unit dialog from this page.

   Click Finish. The IDE creates the “customerdb” package and creates entity classes in this package.

6. Right-click the package that contains the entity classes and choose New > RESTful Web Services from Entity Classes. In the New RESTful Web Services from Entity Classes wizard, click Add All. After the entity classes appear in the Selected Entity Classes field, click Next. The Generated Classes panel opens. Accept the default package names and click Finish.
7. A dialog box opens and asks you how you want to register the resource classes. You can have the IDE register them in the EE 5 style, in a servlet adaptor in the web.xml deployment descriptor. You can have the IDE register them in the EE 6 style, by implementing a subclass of javax.ws.rs.core.Application. Or you can choose to not have the IDE register the resource classes, in which case you have to register them manually, either in a servlet adaptor or through implementing javax.ws.rs.Application.
8. Click Finish. The IDE creates the entity classes and then the RESTful web services.

When you are finished, open the service folder that the IDE just created. This folder contains the resource classes of the service. Note that these classes are all stateless session beans, taking advantage of EE 6.

Samples

The MessageBoard example contains EJB's exposed as RESTful web services, made possible in EE 6. The sample also has a simple JavaScript client containing RESTful client stubs, and performing GET, PUT, POST, and DELETE functions. MessageBoard and other RESTful service samples are available in the IDE, in New Project > Samples > Web Services.

The Project Jersey main page (http://wikis.sun.com/display/Jersey/Main;jsessionid=56A514DBFBA293ADE3B64BF91A781A18) contains links to a number of samples that work with NetBeans. However, the samples are only EE5 and have not been updated since NetBeans IDE 6.5.

JAX-WS Web Services

Creating an Empty JAX-WS Web Service

To create an empty web service or to expose an existing session enterprise bean, perform the following steps:

1. Right-click the project node for a web application or an EJB module and select **New > Other**.
2. Choose **Web Services** from the Categories pane and **Web Service** from the File Types pane.
3. Click the **Next** button.
4. On the next screen, provide a name for the web service. This name is used as the base name for the files created.
5. Specify the location and package for the web service files.
6. Do one of the following:
   - Select the **Create an Empty Web Service** radio button if you want to start with the Java code.
   - Select the **Delegate to Existing Session Enterprise Bean** radio button if you have a session bean you want to expose as an EJB endpoint web service. Click the **Browse** button and navigate to the session facade and select its node.
7. Click the **Finish** button to have the IDE create the web service.

Figure 3-7 illustrates the new web service dialog box.

The IDE creates an implementation class for the web service with the required web service annotations and import statements.

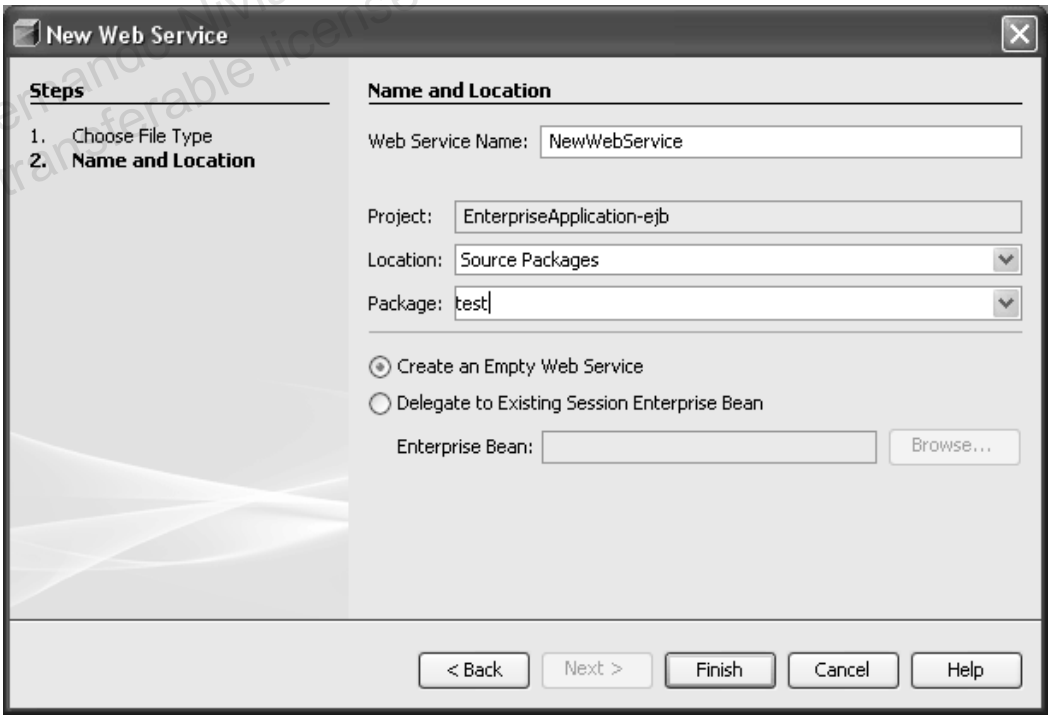


Figure 3-7 The New Web Service Dialog

Creating WSDL Files for JAX-WS Web Services

To create a WSDL file, perform the following steps:

1. Right-click the project node for a web application or an EJB module and select **New > Other**.
2. Choose **XML** from the Categories pane and **WSDL** from the File Types pane.
3. Click the **Next** button.
4. On the next screen, type a name for the WSDL file and specify a location folder.
5. Specify the namespace of the elements defined in the WSDL file.
6. Specify any necessary XML schemas.
7. Click **Finish**.

You can use the IDE to generate a web service or web service client from the WSDL file.

Creating a JAX-WS Web Service From a WSDL File

To create a web service from a WSDL file, you can specify either the local WSDL file or the URL for the WSDL file on the network.

To create a web service from a WSDL file, perform the following steps:

1. Right-click the project node for a web application or an EJB module and select **New > File/Folder**.
2. Choose **Web Services** from the Categories pane and **Web Service from WSDL** from the File Types pane.
3. Click the **Next** button.
4. On the next screen, provide a name for the web service. This name is used as the base name for the files created.
5. Specify the location and package for the web service files.
6. Type the name of the WSDL file or click the **Browse** button and navigate to the WSDL file. Specify the port, if necessary.
7. Click the **Finish** button to have the IDE create the web service.

The IDE creates an implementation class for the web service and when you build the application, the IDE uses the `wsimport` tool to generate the required items for the WSDL file.

Adding Operations to a JAX-WS Web Service

To add an operation to a web service, perform the following steps:

1. Open the web service class in the source editor.
2. In the source editor, right-click in the web service class and select **Web Service > Add Operation** from the menu.
3. In the Add Operation dialog box, provide the method name, return type, parameters and any exceptions you want the method to throw.
4. Click **OK** to close the dialog box.

The wizard provides `java.rmi.RemoteException` as an exception to all operations created on a web service.

Refreshing a JAX-WS Web Service

If you want to refresh the web service, expand the Web Services node in your project. Right-click the web service node and select **Refresh Service**. In the resulting dialog box, update the WSDL as necessary and click **OK**. When you rebuild the application, the code is regenerated.

Editing Web Service Attributes

A web service attribute is a property in a Java class generated from a WSDL file. If your web service is created from a WSDL file, or if you are working with a web service client, you can use the IDE's Web Service Attributes Editor to modify attributes. For example, if you do not like the name of the web service that was generated from the WSDL file, you can use this editor to rename the web service. The IDE regenerates the web service, overriding the name provided by the WSDL file, and synchronizing all related files.

To edit a web service attribute, perform the following steps:

1. In the Projects window, expand the web services node in you web application or EJB module project.
2. Right-click the web service node and select **Edit Web Service Attributes** from the menu to open the window where you can edit the web service attributes.
3. In the web service attributes editor window, expand the section containing the attribute you want to edit. Attributes are grouped by type into sections. You can expand and collapse these sections by clicking on the **+** or **-** symbol.

4. Expand the attribute you want to edit by clicking on the **+** or **-** symbol and modify the attribute as needed.
5. Click **OK** to close the window.

Testing Web Services

The GlassFish Server comes with a tester application enabling you to test your JAX-RS or JAX-WS web service. After you deploy your application, you can test the web service by performing the following steps:

1. Expand the Web Services node in your Web module or EJB module.
2. Right-click the web service and select **Test Web Service**.

The IDE opens the default web browser to a page with a form that allows you to test the web service.

Note – The tester application is only available with the GlassFish Server.

Creating Web Service Clients

To create a web service client, perform the following steps:

1. Right-click the project node for a web application or an EJB module and select **New > Other**.
2. Choose **Web Services** from the Categories pane and **Web Service Client** from the File Types pane.
3. Click the **Next** button.
4. On the next screen, specify the WSDL file of the web service. The NetBeans IDE can create a web services client from a WSDL file obtained from a running web service or from a local file. Select one of the following:
   - Project – Generates a client from a project on your local filesystem. Click **Browse** to locate a project's web service port icon.
   - Local File – Generates a client from a WSDL file on your local filesystem. Type the name of the local file or click **Browse** to locate the WSDL file on your local filesystem.

- WSDL URL – Generates a client from a running web service. Type the URL for the web service. If you are behind a firewall, click **`Set Proxy`** and set your proxy host and port number. The WSDL file is downloaded when you finish the wizard.

5. Specify the package for the client.
6. Click **`Finish`**.

Calling a Web Service Operation

To call a web service operation from a java class or JSP file, perform the following steps:

1. Open the file from which you want to call the web service.
2. In the source editor, right-click in the method (for Java files) or anywhere in the source editor (for JSP file) from where you want to call the web service, and choose **`Web Service Client Resources > Call Web Service Operation`** from the menu.
3. The Select Operation to Invoke dialog box appears. Expand the nodes and select a web service operation.
4. Click **`OK`** to close the dialog.

The IDE adds the code required for calling the web service to the file.

Refreshing Web Services and Clients

If you want to refresh the WSDL file that a client uses, expand the Web Service References node in your client project. Right-click the web service client node and select **`Refresh Client`**.

Message Handlers

The NetBeans IDE supports configuring and using message handlers on the client side, server side, or both. Message handlers inject code into a web service interaction without changing the code. For example, you can use a message handler for adding security to the communication between a web service and client.

Creating Message Handlers

To create a message handler, perform the following steps:

1. Right-click the project node for a web application or an EJB module and select **New > Other**.
2. Choose **Web Services** from the Categories pane and **Message Handler** from the File Types pane.
3. Click the **Next** button.
4. On the next screen, type a name for the message handler and specify a location and package.
5. Click **Finish**.

In the Source Editor, modify the default message handler to suit your requirements.

Configuring Message Handlers

You can configure message handlers on either the service or the client side. In the Projects window, do one of the following: If you are configuring message handlers on the service-side, expand the Web Services node, right-click the node for the web service and choose **Configure Handlers**. If you are configuring message handlers on the client-side, right-click the node for the web service and choose **Configure Handlers**. In the Configure Message Handlers dialog box, click **Add** and browse to the message handler class. Click **OK**. The message handler class is listed in the dialog box. Click **OK** to complete the configuration of the message handler.

Deployment Descriptor Settings

Servlet web service endpoints use the standard `web.xml` deployment descriptor file to configure the servlet. The URL pattern entry for the servlet determines the URL for the web service. If you change this, you must also change the Endpoint Address URI setting in the `sun-web.xml` deployment descriptor.

To modify the Endpoint Address URI setting in the `sun-web.xml` deployment descriptor, perform the following steps:

1. In the Projects window, double-click the `sun-web.xml` file node under the Configuration Files node of your web application project to open the deployment descriptor in the visual editor.
2. Expand the Web Services tab of the visual editor and select the web service.

3. Type the new Endpoint Address URI and save the change.

Module 4

Server Resources

Objectives

Upon completion of this module, you should be able to:

- Use the NetBeans IDE to deploy applications to Java EE application servers
- Use the NetBeans IDE to create a database

Java EE Application Servers

The NetBeans IDE supports deploying applications to the GlassFish Server, BEA Weblogic Application Server, JBoss Application Server, and Tomcat.

You can view server resources in the Services window in the NetBeans IDE.

Registering Application Servers

To register an application server:

1. Select **Servers** from the Tools menu.
2. Click the **Add Server** button to start the Add Server Instance wizard.
3. In the wizard, choose the server type and provide a logical name. Click the **Next** button.
4. For the GlassFish server, provide the following:
   a. Platform Location – The installation directory of the application server
   b. Register Default Local Domain – Registers the default domain created during the installation of the application server. The next screen asks for the administration user name and password.
   c. Register Local Domain – Registers a local domain. The next screen asks for the location of that domain. The following screen asks for the user name and password of that domain.
   d. Register Remote Domain – Registers an application server running on a different machine. The next screen asks for the host name and port number of that instance. The following screen asks for the administration user name and password.
   e. Create Personal Domain – Creates and registers a new local domain. The next screen asks for the location for the new domain. The following screen asks for the administration user name and password for that domain. The final screen allows you to specify ports for the new domain.
5. Click the **Finish** button to register the application server.

The server now appears in the Servers window.

Starting the Application Servers

To start an application server:

1. Switch to the Services window
2. Expand the Servers node.
3. Right-click on the server and select **`Start`** from the contextual menu.

The server starts and its log file(s) open in the output widow.

Examining Application Servers

You can examine the GlassFish Server in two ways:

- Using the NetBeans IDE Services Tab
- Using the Administration GUI

You can use the Services tab within the NetBeans IDE to examine the applications and resources that are deployed to the application server. For each node, you can right-click and select Refresh to see the current state. For applications, you can use the Services tab in the IDE to undeploy a Java EE application or module. Most of the information presented within the IDE's view of the application server is read-only. In other words, to make changes on the application server you need to redeploy resources or use the Administration GUI.

The Administration GUI for the application server is used to configure, examine, and modify resources within the application server. It is a web application, so you need a web browser to access it.

To administer the application server running on your workstation, perform the following steps:

1. Switch to the Services window
2. Expand the Servers node
3. Right-click the server node in the Services window and select **`View Admin Console`**.

4. The URL **`http://localhost:4848`** opens in your web browser. The port 4848 is the default port specified during installation. Your administration port might be different.
5. Provide the administration user name and password (defaults are **`admin`**/**`adminadmin`**) on the welcome screen. Upon successful login, you should see the screen illustrated in Figure 4-1.

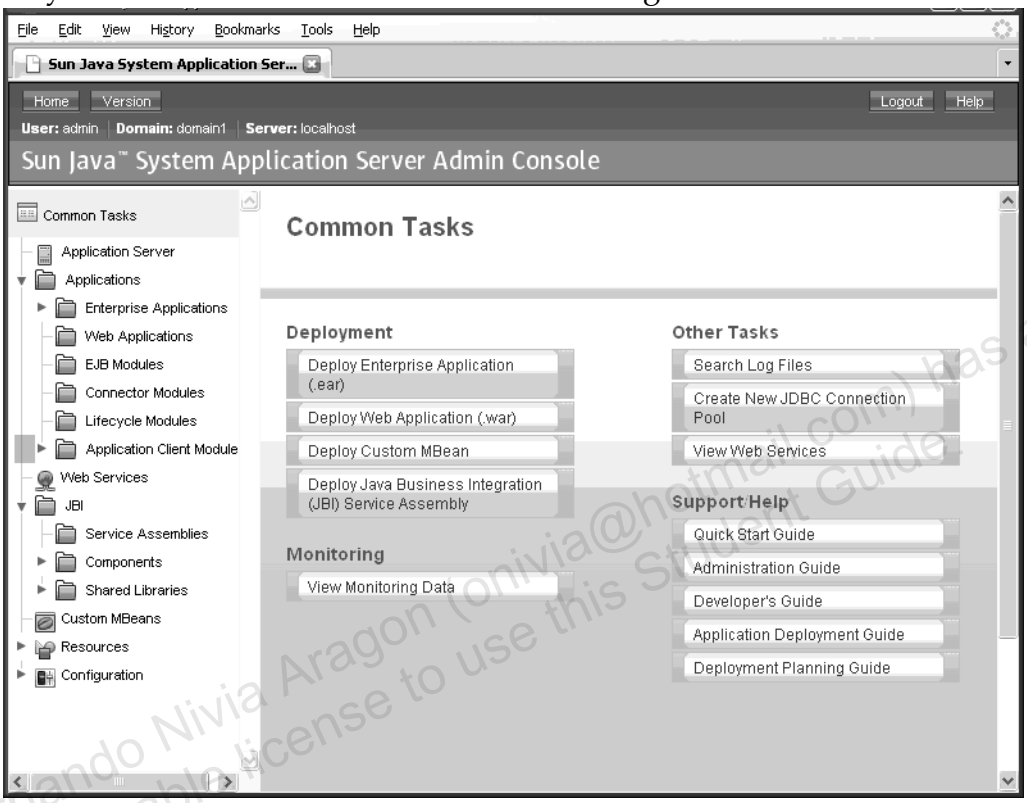


Figure 4-1 The GlassFish Server Administration Console

Configuring Application Servers

To configure the GlassFish Server, log in to the administration console. To connect to the console, open the URL **`http://localhost:4848`** in your web browser and log in with the administration user name and password (default: **`admin`**/**`adminadmin`**). The application server configuration information can be found under the Configuration node.

ORB Settings

In the administration console, expand the following: **`Configuration > ORB > IIOP Listeners`** node. Select the **`orb-listener-1`** node. The IIOP listener port can be viewed in the right pane.

Stopping Application Servers

To stop the application server:

1. Switch to the Services window of the IDE.
2. Expand the Servers node.
3. Right-click on the server and select **Stop** from the contextual menu.

A message in the Output window is displayed that indicates that the server has been shut down.

Examining Server Log Files

To view the server log file, expand the Servers node in the Services window, right-click the application server node, and select **View Server Log**. The GlassFish Server log appears in the output window. The log file (`server.log`) can be found in the *JAVAEE_HOME*`/domains/domain1/logs` directory.

Classpath Settings

To configure the classpath settings for the application server, perform the following steps:

1. Log into the administration console.
2. Select the **Application Server** node in the left pane.
3. In the right pane, select the **JVM Settings** tab.
4. Select the **Path Settings** tab in the JVM Settings tab.
5. In the text areas on this page, you can modify the server's classpath.
6. Click the **Save** button to save any changes you make.

You need to restart the application server for your changes to take effect.

Adding JVM Machine Options

To add JVM machine options for the application server, perform the following steps:

1. Log into the administration console.
2. Click the Application Server node in the left pane.
3. In the right pane, click the JVM Settings tab.
4. Click the JVM Options tab in the JVM Settings tab.
5. Click the Add JVM Option button to create a new text field in the JVM options table.
6. In the new text area, type the new JVM option.
7. Click the Save button to save any changes you make.

You need to restart the application server for your changes to take effect.

You can remove the JVM options by selecting the checkbox for the option in the JVM options table and clicking the Delete button. Click the Save button to save any changes you make and restart the application server.

Modifying SOAP Message Security Settings

To modify the default provider for server-side and client side Simple Object Access Protocol (SOAP) message security, perform the following steps:

1. Log in to the administration console.
2. In the left pane, expand the Configuration node, then expand the Security node, the Message Security node and click the SOAP node.
3. In the Message Security tab in the right pane, select the default provider or default client provider from the drop-down list.
4. Click the Save button to save any changes you make.

You must restart the application server for your changes to take effect.

Administering Security

You can administer the security configuration of the application server using the administration console. To connect to the console, open the URL **`http://localhost:4848`** in your web browser and log in with the administration user name and password (default: **`admin`**/**`adminadmin`**).

Adding a File Realm User

In the administration console, expand the following: **`Configuration > Security > Realms`** and select the **`file`** realm. In the right pane of the web browser, click the **`Manage Users`** button. The pane lists the current users in the file realm. Click the **`New`** button to create a new user in the file realm. In the form, provide the user ID, password (twice) and the comma-delimited group list. Click the **`OK`** button at the top of the frame to add the user.

Administering the Java Message Service

You can administer the Java Message Service (JMS) resources using the GlassFish Server administration console. To connect to the console, open the URL **`http://localhost:4848`** in your web browser and log in with the administration user name and password (default: **`admin`**/**`adminadmin`**).

Creating Physical Destinations

In the administration console, expand the Configuration node in the left pane. Expand the Java Message Service node and select the Physical Destinations node. In the right pane, click the **`New`** button. Specify a name for the physical destination and its type (topic or queue). Click the **`OK`** button to create the physical destination.

Deploying to a Server

You can deploy assembled WAR, EJB-JAR, and EAR files to the application server using the administration console.

Deploying WAR Files

In the administration console, expand the **Applications** node and select the **Web Applications** node. Click the **Deploy** button to deploy a WAR file. In the form, click the **Browse** button to navigate to the WAR file you want to deploy. Click the **Open** button. The WAR file appears in the text field. Click the **OK** button at the top of the frame to deploy the WAR file.

Deploying EJB JAR Files

In the administration console, expand the **Applications** node and select the **EJB Modules** node. Click the **Deploy** button to deploy an EJB JAR file. In the form, click the **Browse** button to navigate to the JAR file you want to deploy. Click the **Open** button. The JAR file appears in the text field. Click the **OK** button at the top of the frame to deploy the EJB JAR file.

Deploying EAR Files

In the administration console, expand the **Applications** node and select the **Enterprise Applications** node. Click the **Deploy** button to deploy an EAR file. In the form, click the **Browse** button to navigate to the EAR file you want to deploy. Click the **Open** button. The EAR file appears in the text field. Click the **OK** button at the top of the frame to deploy the EAR file.

Databases

You can use the NetBeans IDE to administer the Java DB database.

Starting the Java DB Database

To start the Java DB database, select **Java DB Database > Start Server** from the Tools menu.

The Java DB Database Process in the Output window indicates that the database is running and ready to accept connections.

Creating a Java DB Database

To create a new Java DB database, perform the following steps:

1. Ensure that the Java DB database server is started.
2. Select **Java DB Database** > **Create Database** from the Tools menu.
3. Provide the following information in the dialog box:
   a. Database Name – Provide the name for the database. The name should not contain spaces.
   b. User Name – Provide the user name needed to connect to the database.
   c. Password – Provide the password needed to connect to the database.

Note – It is best to avoid using user name and password that are SQL keywords.

4. Click the **OK** button to create the database.

A Java DB database instance is created in the location specified. A connection to the new database is created under the Databases node in the Services window.

Modifying the Java DB Database Location

To modify the location for the Java DB databases, perform the following steps:

1. Select **Java DB Database** > **Settings** from the Tools menu.
2. Provide the following information in the dialog box:
   a. Java DB Installation – Accept the location of the Java DB installation.
   b. Database Location – Provide the location of the database folder to specify the new base directory for Derby database instances.
3. Click the **OK** button to save your changes.

You need to copy any existing databases to the new location and restart the Java DB database server to access those databases.

Stopping the Java DB Database

To stop the Derby database, select **Java DB Database** > **Stop Server** from the Tools menu.

The Java DB Database Process window indicates that the database has been shut down.

Adding JDBC Drivers

You can add JDBC drivers for any compliant database server to the IDE. In the Services window, expand the Databases node. Right-click the Drivers node and select **New Driver**. You must specify the driver files and driver class and provide a name for the driver within the IDE. When added, the driver appears as a node under the Drivers node.

Connecting to Databases

To connect to a database, first ensure that the database server is running. In the Services window, expand the Databases node, then expand the Drivers node. Right-click the node for the driver you want to use and select **`Connect Using`** from the menu. Specify the driver name, database URL, user name and password. Click the **`OK`** button to establish the connection. Figure 4-2 illustrates the database connection dialog.

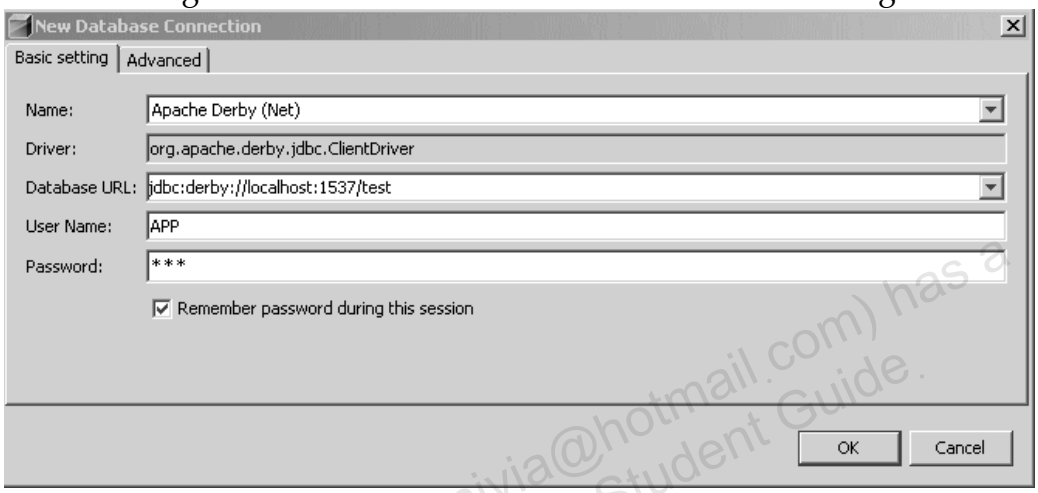


Figure 4-2 The Database Connection Dialog

The IDE creates a persistent node under the Databases node in the Services window for this connection.

Interacting with Databases

To examine the contents of a database, expand the corresponding connection node under the Databases node in the Services window. Under the connection node, you see nodes for the tables, views, and procedures in the database.

To quickly view the data in an SQL table, right-click the appropriate table node and select View Data. A NetBeans IDE SQL command window opens and executes an SQL command to display the data in the table in the window.

Executing SQL Queries

To execute a SQL query in the NetBeans IDE, expand the Databases node in the Services window, right-click the appropriate connection and select **Execute Command**. The NetBeans IDE SQL command window opens. Enter the SQL query you want to execute, and click the **Run SQL** button in the toolbar (or press **Ctrl-Shift-E**).

Capturing Database Schemas

To create a new database schema, right-click your web application or EJB module project node and select **New > Other**. Select **Persistence** from the Categories pane and **Database Schema** from the File Types pane. Click the **Next** button to continue. Provide a file name for the schema. Specify the folder in the project for the schema. Click the **Next** button to continue. On the next screen, select the database connection that is used to create the schema. Click the **Next** button to continue. On the next screen, select the tables in the left pane and click the **Add** button to add them to the schema. To add all the tables, click the **Add All** button. To remove tables, select them in the right pane and click the **Remove** button. When the right pane contains the tables you want in the schema, click the **Finish** button. The IDE creates the schema in the project. A node for the schema is created under the Configuration Files node.

Appendix A

NetBeans IDE 6.8 Keyboard Shortcuts

This appendix contains NetBeans 6.8 keyboard shortcuts. These are available in the NetBeans IDE by selecting the Keyboard Shortcuts Card from the Help menu. You must specify the location of your PDF viewer to see the document.

Keyboard Shortcuts

Highlights of NetBeans IDE 6.8 Keyboard Shortcuts & Code Templates

Finding, Searching, and Replacing

Ctrl-F3	Search word at insert point
F3/Shift-F3	Find next/previous in file
Ctrl-F/H	Find/Replace in file
Alt-F7	Find usages
Ctrl-Shift-F/H	Find/replace in projects
Alt-Shift-U	Find usages results
Alt-Shift-H	Turn off search result highlights
Ctrl-R	Rename
Ctrl-U, then U	Convert selection to uppercase
Ctrl-U, then L	Convert selection to lowercase
Ctrl-U, then S	Toggle case of selection
Ctrl-Shift-V	Paste formatted
Ctrl-I	Jump to quick search field

Navigating through Source Code

Ctrl-O/Alt-Shift-O	Go to type/file
Ctrl-Shift-T	Go to JUnit test
Alt-O	Go to source
Ctrl-B	Go to declaration
Ctrl-G	Go to line
Ctrl-Shift-M	Toggle add/remove bookmark
Ctrl-Shift-Period/Comma	Next/previous bookmark
Ctrl-Period/Comma	Next/previous usage/compile error
Ctrl-Shift-1/2/3	Select in Projects/Files/Favorites
Ctrl-[	Move caret to matching bracket
Ctrl-K/Ctrl-Shift K	Next/previous word match
Alt-Left/Alt-Right/Ctrl-Q	Go backward/forward/to last edit
Alt Up/Down	Next/previous marked occurrence

Coding in Java

Alt-Insert	Generate code
Ctrl-Shift-I	Fix all class imports
Alt-Shift-I	Fix selected class's import
Alt-Shift-F	Format selection
Alt-Shift Left/Right/Up/Down	Shift lines left/right/up/down
Ctrl-Shift-Up/D	Copy lines up/down
Ctrl/Alt-F12	Inspect members/hierarchy
Ctrl-/	Add/remove comment lines
Ctrl-E	Delete current line

Coding in C/C++

Alt-Shift-C	Go to declaration
Ctrl-F9	Evaluate expression

Coding in Ruby

Ctrl-Shift-A	Jump Rails action > view
Alt-Shift-Period/Comma	Select Next/Previous element
Ctrl-Shift-Space	Show documentation
Ctrl-Shift-T	Jump from test file to file

Compiling, Testing, and Running

F9	Compile package/ file
F11	Build main project
Shift-F11	Clean & build main project
Ctrl-Q	Set request parameters
Ctrl-Shift-U	Create JUnit test
Ctrl-F6/Alt-F6	Run JUnit test on file/project
F6/Shift-F6	Run main project/file

Opening and Toggling between Views

Ctrl-Tab (Ctrl-`)	Toggle between open documents
Shift-Escape	Maximize window (toggle)
Ctrl-F4/Ctrl-W	Close currently selected window
Ctrl-Shift-F4	Close all windows
Shift-F10	Open contextual menu
Alt-Shift-D	Undock window

Debugging

Ctrl-F5	Start debugging main project
Ctrl-Shift-F5	Start debugging current file
Ctrl-Shift-F6	Start debugging test for file (JUnit)
Shift-F5/F5	Stop/Continue debugging session
F4	Run to cursor location in file
F7/F8	Step into/over
Ctrl-F7	Step out
Ctrl-Alt-Up	Go to called method
Ctrl-Alt-Down	Go to calling method
Ctrl-F9	Evaluate expression
Ctrl-F8	Toggle breakpoint
Ctrl-Shift-F8	New breakpoint
Ctrl-Shift-F7	New watch

Complete List of Keyboard Shortcuts:

http://wiki.netbeans.org/KeymapProfileFor60

Appendix B

Java EE Annotation Reference

This appendix contains a summary of Java EE annotations.

Resource Annotations

@Resource(name="", resourceType="",
AuthenticationType=[*CONTAINER*,Application], shareable=true,
jndiName="")

@Resources(Resource[] value)

@EJB(name="", businessInterface="", jndiName="")

@EJBs(EJB[] value)

@Home(name="", homeInterface="")

@Inject(jndiName="")

EJB Annotations

The following contains the bean-type annotations.

Bean-Type Annotations

@Stateless(name="")

@Stateful(name="")

@MessageDriven(name="", ActivationConfigProperty[]={})

@Entity(name="", access=[*PROPERTY*, FIELD], entityType=[*CMP*, BMP], version=3)

@Remote

@Local

@BusinessMethod

Transaction and Security Annotations

@TransactionManagement(value=[*CONTAINER*, BEAN])

@TransactionAttribute(value=[*REQUIRED*, REQUIRES_NEW, MANDATORY, SUPPORTS, NOT_SUPPORTED, NEVER])

@MethodPermissions(String [] roles)

@SecurityRoles(String [] rolesAllowed={}, roleReferences={})

@Unchecked

@Exclude

@RunAs(String rolename)

@SecurityRoles(String [] RoleNames={})

Callback Annotations

@CallBackListener(String classname)

@PostConstruct

@PreDestroy

@PostActivate

@PrePassivate

@EntityListener(String classname)

@PrePersist

@PostPersist

@PreRemove

@PostRemove

@PreUpdate

@PostLoad